



# DEPARTMENT OF THE NAVY XML DEVELOPER'S GUIDE

Department of the Navy  
Office of the Chief Information Officer  
XML Work Group  
29 October 2001



**DONXML WG – TechTeam**

# **Initial DON XML Developer's Guide**

**This version:**

DONXML Developer's Guide – 29 October 2001

**Latest version:**

DONXML Developer's Guide - 29 October 2001

**Previous version:**

Summary DON XML Developer's Guide - Final Draft 25 October 2001

**Editor:**

Brian Hopkins ([bhopkins@logicon.com](mailto:bhopkins@logicon.com))

---

## **About This Document**

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the [NavyXML Quickplace](#)<sup>i</sup>. Additional DON XML policy and guidance can also be found at the NavyXML Quickplace.*

This document represents an abbreviated version of the full 9 October Consensus Draft titled, “XML Developers Guide – 9 October”, it did not go through the full consensus process as described by DONXML TechTeam [Operating Guidelines](#) and therefore does not represent a consensus of the entire team. This document was produced by key individuals of the DONXML Technical Team and Steering Group in order to support the Task Force Web (TFWeb) pilot project deadlines and milestones.

This document is an early deliverable of the overall DONXML strategy for employing XML within the department. It is intended to provide general development guidance for the many XML initiatives currently taking place within the DON. In the mean time, the DONXML Work Group (WG) is in the process of developing a long-term strategy for aligning XML implementations with the business needs of the department. This version of the guidance is primarily written to assist developers in creating schemas that describe XML [payloads](#) of information. It should be noted that payloads represent only one component required for secure, reliable information exchange. Other components include a specification for reliable messaging (including authentication, encryption, queuing, and error handling), business service registry and repository functions, and

transport protocols. Emerging technologies and specifications are, or will shortly, provide XML-based solutions to many of these needs. The DONXML WG is developing an XML Primer that will describe each of these components and bring together the overall strategy for capitalizing on XML as a tool for enterprise interoperability.

This Developer's Guide will be a living document, which will be updated frequently. Future releases of this document will represent full DONXML TechTeam consensus.

Paragraphs of this document are broken into three parts.

- "Guidance" provides a concise summary of requirements and recommendations.
- "Explanation" provides a brief explanation of the reasoning behind the guidance provided.
- "Example" provides one or more examples pertaining to the guidance.

The bulk of this document is contained in appendices that are provided as non-[normative](#) supplementary information. The appendices should be considered to have a "draft" status, and do not represent the consensus of the DONXML TechTeam.

This document is primarily intended for developers already familiar with XML; however it has a comprehensive glossary, which provides good starting points for XML beginners. Some of this document focuses on XML Schemas as a tool for interoperability. To get the maximum benefit, it is suggested that you take the time to become familiar with the [XML Schema](#) language. An excellent tutorial with labs is available at <http://www.xfront.com/>.

We encourage developers to try the techniques recommended here and provide feedback to the DONXML TechTeam via the [editor](#). We will collect lessons learned and best practices, updating and expanding this document periodically.

---

## Table of Contents

1.	REFERENCES .....	3
2.	INTRODUCTION .....	3
3.	INDICATING REQUIREMENTS AND RECOMMENDATIONS .....	4
4.	TERMINOLOGY.....	4
5.	RECOMMENDED XML SPECIFICATIONS .....	4
6.	XML CONVENTIONS .....	5
6.1.	XML Components .....	5
6.1.1.	Standardized Case Convention.....	5
6.1.2.	Usage of Acronyms and Abbreviations .....	6
6.1.3.	XML Component Naming .....	7
6.2.	Schema Design.....	9

**Initial XML Developer's Guide - 29 October 2001**

6.2.1.	Schema Languages .....	9
6.2.2.	Recommended Schema Development Methodology .....	10
6.2.3.	Capturing Metadata.....	12
<b>6.3.</b>	<b>Document Annotations.....</b>	<b>15</b>
6.3.1.	Document Versioning.....	15
6.3.2.	Headers .....	17
<b>6.4.</b>	<b>Attribute vs. Elements .....</b>	<b>19</b>
<b>7.</b>	<b>COE XML REGISTRY .....</b>	<b>20</b>
<b>8.</b>	<b>POINTS OF CONTACT .....</b>	<b>21</b>
	DONXML WG Government Lead: .....	21
	DONXML TechTeam Lead and Editor: .....	21
<b>9.</b>	<b>APPENDICES .....</b>	<b>1</b>
<b>Appendix A – ebXML and the eBTWG.....</b>		<b>1</b>
	Description.....	1
	ebXML Naming Rules .....	2
	Representation Terms .....	4
<b>Appendix B – Schema Development.....</b>		<b>1</b>
	Possible Schema Development Procedure Summary.....	1
<b>Appendix C - Tools and References .....</b>		<b>1</b>
	Tools .....	1
	Publications .....	1
	Internet .....	2
<b>Appendix D – W3C XML Recommendations .....</b>		<b>1</b>
<b>Appendix E – Combined XML Schema Example.....</b>		<b>1</b>
<b>Appendix F – Sample XML Document Headers.....</b>		<b>1</b>
<b>Appendix G – Draft Glossary and Acronyms.....</b>		<b>1</b>
	Terms .....	1

---

## 1. References

- (a) DON CIO Interim Policy on the Use of Extensible Markup Language For Data Exchange  
dtd 06 Sept 2001

## 2. Introduction

In August 2001 DON CIO established a [DON XML Work Group](#) (DONXML WG) to provide the leadership and guidance to maximize the value and effectiveness of emerging XML component technologies implemented across the DON Enterprise. At its first meeting in August 2001, the DONXML WG agreed to

## Initial XML Developer's Guide - 29 October 2001

produce a DON XML Developer's Guide as a deliverable. This document serves as a reference guide for making existing applications "XML-enabled", and for developing future capabilities that will leverage XML to the maximum extent possible.

Service initiatives such as [Task Force Web](#)<sup>ii</sup> (TFWeb) are implementing XML enabled applications very quickly. This document will assist DON activities in developing XML implementations in the short term, while lessons learned are collected.

On 6 September 2001 the Department of the Navy, Chief Information Officer signed out reference (a), an Interim XML Policy Statement on the use of XML within the department. Copies of this policy are available on the NavyXML QuickPlace.

## 3. Indicating Requirements and Recommendations

The terms "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" are used throughout this document, and should be interpreted in accordance with [RFC 2119](#)<sup>iii</sup>.

Most items in this document should be considered as guidance rather than requirements. Specifically, items using the terms "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", and "**SHALL NOT**" should be considered as the only requirements.

DON CIO understands that short timeframe XML implementations (such as TFWeb), or pre-existing schema that do not follow this guidance, cannot be changed immediately. Activities should read this document and develop a migration plan to evolve their current XML implementations; additionally, the DONXML TechTeam encourages submission of feedback as lessons learned are collected.

This document has been reviewed by TFWeb engineers and is intended to extend/compliment direction provided by that effort. Please bring any possible inconsistencies or suggestions to the attention of the DONXML WG and the [editor](#).

## 4. Terminology

The term [XML](#) is used to describe a large range of specifications and technologies associated with XML [markup](#). It is critical that activities developing XML-enabled applications have a firm understanding of basic XML terminology. [Appendix G](#) provides a list of applicable acronyms and terms.

Many schema languages have been created for expressing validation rules; however, throughout this document the term 'schema' with a small 's' is used generically (to include DTDs), while the term XML Schema or just Schema (capital 'S') refers specifically to schemas authored in accordance with the W3C XML Schema recommendation.

## 5. Recommended XML Specifications

### Guidance

In general, production applications **SHOULD** only use software that implements [W3C Final Recommendations](#). Applications using software that implement W3C specifications at other stages of the development process, or non-W3C specification, **MUST** do so with the following restrictions:

- W3C Proposal Recommendations – Suitable for production implementation with the understanding that any changes in the specification, upon final recommendation, may require recoding or retesting.
- W3C Candidate Recommendations – Suitable for pilot implementations.
- W3C Working Drafts – Suitable for prototypes and Advance Concept Demonstrations.
- Other "Final Specifications" from credible organizations (OASIS, UN/CEFACT, ebXML, ANSI, ISO...) – Suitable for pilot and possibly production implementations. Example, [SAX](#)

**Initial XML Developer's Guide - 29 October 2001**

- Other working/draft specifications - Suitable for prototypes and Advance Concept Demonstrations

**Other guidance:**

- All XML parsers, generators, validators, enabled applications, servers, databases, operating systems, and other software acquired or used by DON activities **SHALL** be fully compliant with all W3C XML specifications holding recommended status.
- Proprietary XML or XSL syntax extensions, beyond those included in W3C or DON recommended specifications addressed in this document:
  - **MUST NOT** be employed for any use of XML or XSL that is shared publicly with activities outside a particular development environment.
  - **SHOULD** only be employed privately (within a homogeneous development environment) after careful evaluation of possible impacts on cross-platform interoperability, and dependency on software from a single vendor. This decision **MUST** only be made by government program managers.

**Explanation**

[Appendix D](#) provides a list of W3C recommendations and their current stage in the consensus process (as of the date of this document). The following table provides a list of XML specifications or standards that are not W3C products. Two categories are provided. The "Recommended" column represents widely adopted standards that are believed to be mature and uniformly supported by software implementations. The "Maturing" column represents other standards that the DONXML WG believes to be sufficiently mature, however they may not be uniformly supported in existing software implementations, so caution is advised. Future versions of this document will add additional specifications from other standards bodies and efforts such as ebXML, OASIS, UN/CEFACT, etc.

<i>Recommended</i>	<i>Maturing</i>
<a href="#">SAX</a> <sup>1</sup> 1.0 and 2.0	<a href="#">SOAP 1.1</a> (W3C Note)

Application vendors often provide proprietary extensions to adopted standards. These extensions often simplify the job of software developers, but they also make developed systems dependant on software from a single vendor, and often they also restrict the software to being run on a single vendor's operating system or hardware. Government program managers must have the final say in the decision to employ such extensions, even when doing so inside a single system's boundaries or within a homogeneous development environment.

## 6. XML Conventions

### 6.1. XML Components

#### 6.1.1. Standardized Case Convention

---

<sup>1</sup> SAX is not a specification developed by a standards body or the W3C. It is an open source project maintained by a community of developers. SAX parsers have been written for several languages, but the only platform independent version is the Java API. A parser that is SAX compliant must implement an equivalent to the Java API, which is provided at the SAX homepage.

## Initial XML Developer's Guide - 29 October 2001

### Guidance

DON developers **SHALL** adopt the [camel case](#) convention, as defined by the [ebXML](#) Technical Architecture, when creating [XML component](#) names. Excerpts are provided in [Appendix A](#).

- [XML Elements](#) and [XML Schema Types](#) use *upper camel case*: The first letter in the name is upper case, as is the letter beginning each subsequent word.
- [XML Attributes](#) use *lower camel case*: Like upper camel case, except the first letter of the first word is lower case.

### Explanation

Major XML consortia such as OASIS, UN/CEFACT, RosettaNet, Biztalk and ebXML (see Internet references in [Appendix C](#)) have all adopted the camel case convention for [XML component](#) naming, with ebXML differentiating between upper and lower camel case.

### Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Example of an upper camel case element and lower camel case attribute
-->
<UpperCamelCaseElement
    lowerCamelCaseAttribute="foo" />
```

## 6.1.2. Usage of Acronyms and Abbreviations

### Guidance

DON developers **SHALL** follow the ebXML guidance for usage of acronyms or abbreviations in [XML component](#) names:

- Abbreviations **MUST** not be used.
- Acronyms **SHOULD** be avoided, but when used **MUST** be in upper case regardless of their position in [XML component](#) names or the [camel case](#) convention.
- Commonly used acronyms **MAY** be used; however, the decision to use an acronym in a component name **SHALL** be made by program and functional manager rather than by application developers
- Acronyms used in component names **MUST** be spelled out in the component definition that is required to be included via schema annotations (as [XML comments](#) or inside [XML Schema annotation](#) `<xsd:documentation>` elements) (see [Section 6.2.3.2](#)).

### Explanation

XML [documents](#) that rely heavily on terse abbreviated component names are difficult to understand, and subject to misinterpretation. The general consensus among the major XML standards development consortia is that abbreviations should be avoided and acronyms used sparingly.

**Example:**

Example of providing an element definition in a DTD. Note that the acronym DOD is spelled out in the definition.

```
<!-- DODActivityAddressCode  
Definition: A 6-digit code used to uniquely identify organizations within  
the Department of Defense (DOD)  
-->  
<!ELEMENT DODActivityAddressCode (#PCData)>
```

### 6.1.3. XML Component Naming

**Guidance**

The selection of [XML component](#) names **MUST** be a thoughtful process involving business, functional, database, and system subject matter experts. In the [schema](#) design process, DON XML developers **MAY** use temporary or dummy XML component names while consensus is being reached on more carefully designed and defined names.

In accordance with reference (a), components registered with the [COE XML registry](#) **MUST** be reused if available for [XML element](#) names and element domain restrictions.

When tags cannot be found in the registry that are suitable for reuse as XML elements, or when creating attribute or XML Schema types, DON XML developers **SHOULD** create XML component names using the [ISO 11179](#) rules as modified by ebXML with the following caveats.

- For [XML Elements](#): Use an ISO 11179 compliant name or a [business term](#) ([section 6.1.3.1](#)), if appropriate, in upper camel case.
- When a business term is used in place of an ISO 11179 name, the ISO 11179 name **SHOULD** be captured in the schema via XML comments or [XML Schema annotations](#).
- For [XML Attributes](#), use an ISO 11179 compliant name in lower camel case.
- For [XML Schema Types](#), use an ISO 11179 compliant name in upper camel case.

When a tag is found in the COE registry, but does not conform to the [camel case](#) convention and is not either [ISO 11179](#) compliant or a commonly used [business term](#), you **SHOULD**:

- If you are developing in Schemas:
  - Create an XML Schema type from the ISO 11179 compliant data element name. You **SHOULD** document the type with metadata from the COE registry such as the definition, URL to the item, and registry identifier. Additionally you **SHOULD** apply any domain restrictions to the type rather than the element.
  - Create an element referencing the above-created type using the COE Registry name.
  - If a business term exists, create a second element referencing the type and declare it to be in the [substitution group](#) of the COE Registry element.
- If developing in DTDs:
  - You should use the business term, if one exists, as the element name.



## Initial XML Developer's Guide - 29 October 2001

- You **MAY** capture the COE Registry name and ISO 11179 element names in [XML comments](#).
- However, you **MUST** reference the COE registry tag name in the comments if you choose to use a more appropriate business term or ISO 11179 name.
- Register newly created tags with the COE registry in the appropriate namespace.

### Explanation

XML allows the flexibility to express complex names in many ways. [ISO 11179](#) has established a draft data element naming convention that has been adopted by [ebXML](#) for definition of data elements (see [Appendix A](#)). It provides a uniform methodology for choosing and ordering words in complex element names.

In summary, an ISO 11179 compliant name consists of three parts:

- An “Object Class” term, which describes the kind of thing being referred to. This Object Class may consist of one or more words, some of which may be context terms.

For example, the ISO 11179 name ‘*AcousticSignalFrequencyMeasure*’ has the “object class” ‘*Acoustic Signal*’.

- A “Property Term” which is the property of the thing being referred to, which may consist of one or more words.

For example, the ISO 11179 name ‘*AcousticSignalFrequencyMeasure*’ has the property term ‘*Frequency*’.

- A “Representation Term” which identifies allowable values for an element. This list is taken from an enumerated list of allowable representation types (see appendix A).

For example, the ISO 11179 name ‘*AcousticSignalFrequencyMeasure*’ has the representation type “Representation Term” ‘*Measure*’.

The ebXML Technical Report, [Naming Convention for Core Components](#) provides 14 “rules” for constructing a proper element names, some considerations are:

- When the Representation Type and the Property Term are redundant, the property term is dropped, so ‘Item. Identification. Identifier’ becomes ‘Item. Identifier’
- When an element describes an entire class of things (e.g., not a specific property of it), the Property Term may again be dropped, for instance ‘Documentation. Identifier’
- An aggregate component shall have a representation type of ‘*details*’.

The ISO 11179 element name is converted to camel case by removing the periods, spaces and adjusting the capitalization. Note that ISO 11179 names **MAY** be made directly into XML element names, but this is not recommended. As discussed in [Section 6.1.3.1](#) and [Appendix B](#), the XML element name **SHOULD** be an understandable [business term](#) if one naturally exists.

The excerpts provided in [Appendix A](#) were taken from draft documents that are evolving rapidly. This information **SHOULD** be used as guidance only, but may prove helpful.

### Example

See [Appendix B](#) and [Appendix E](#).

#### 6.1.3.1. Business Terms

## Initial XML Developer's Guide - 29 October 2001

### Guidance

Developer's **SHOULD** substitute [business terms](#) for ISO 11179 compliant names in [element](#) declarations when appropriate business terms exist, however the underlying ISO 11179 name **SHOULD** be captured:

- By defining a fixed '**type**' attribute in the [schema](#) referencing the ISO 11179 name. This is primarily if developing in DTDs.
- If developing [XML Schemas](#), '[XML Schema types](#)' or [substitution groups](#) **MAY** be used to reference ISO 11179 names.
- When choosing [XML component](#) names and business terms:
  - Involve domain subject matter experts (operational personnel, program managers, etc) and functional data experts (database administrators, functional data manager, data modelers, etc.) Application developer's **MUST NOT** be left on their own in the creation of XML component names.
  - Use existing definitions (from the [DDDS](#), COE Data Emporium, MIL-STDs, or other credible standard data element definitions).
  - Business terms **SHOULD NOT** be created just for the sake of having one; the existence and use of business terms **SHOULD** be determined by consensus of a community of users. When a business term is not apparent or does not exist, the ISO 11179 complaint name **MAY** be used as the XML component name instead.

More than one business term may exist for a single element, such as when an acronym is commonly used instead of the full business name. If developing XML Schemas, extra synonymous business terms **MAY** be created and declared in the substitution group of the primary business term.

### Explanation

The ebXML deliverables define the concept of a [Business Term](#). Business terms are commonly recognized words that are more appropriately used as [XML element](#) names, rather than the often-esoteric [ISO 11179](#) conventions. Business terms improve the readability of [schemas](#) and [instances](#), while the ISO 11179 names provide more precise and structured semantics. Both are desirable when business and technical personnel are working together to define [XML grammars](#) for the exchange of business information by IT systems.

This guidance may appear confusing because on one hand the creation of ISO 11179 names is recommended, but on the other, business terms are recommended for XML element names. The guidance is to define ISO 11179 standard names and capture those names through the use of the Schema "type" while retaining readability through using business terms which show up in the XML instance. Since the XML Schema is XML, those analysts interested in finding out, for instance, that "National Stock Number" is a business term for "***Federal Material Item. Identification. Details***" can look at the underlying type name of the `<NationalStockNumber>` tag.

### Examples

See [Appendix E](#).

## 6.2. Schema Design

### 6.2.1. Schema Languages

#### Guidance

Only W3C recommended languages **SHALL** be used within the DON, for describing [documents](#). Specifically, the [DTD](#) and the [W3C recommended XML Schema](#) language.

All activities developing data-oriented [schema](#) in [DTD](#) syntax **SHOULD** plan on migrating to [XML Schemas](#) in their next software release.

## Initial XML Developer's Guide - 29 October 2001

DON XML developers **MAY** elect to use [DTD's](#) for markup of data that is strictly document-oriented (paragraph, chapter, appendix...), however the XML Schema language is preferred.

### Explanation

The XML Schema language is the DONXML WG recommended method for creating [schemas](#). XML Schemas provide a rich syntax for expressing metadata. Some of its features include:

- Structures are defined, in [Part 1](#) of the XML Schema recommendation, that allow the definition of relational (keyed) data, and object-oriented (type inheritance) data.
- Several flexible options for defining element content models are specified.
- The language also provides for better modularity by allowing two different ways, include or import, to reuse external Schemas.
- [Part 2](#) of the XML Schema recommendation deals with data types. The DTD syntax allows for expression of only a few data types, and only one data type (a string) may be assigned as contents of an [XML Element](#) (the others are for different types of [XML Attributes](#)). XML Schemas have dozens of built-in data types, and allow creating custom data types from combinations of the built-in set.
- The concept of a "type" is extended beyond simple data types (string, Boolean, integer, etc.). Complex types may be declared and named, creating stereotype content models of other XML elements; these can then be extended, restricted, and assigned to XML elements in an "object-oriented" fashion.
- Several means of further constraining element and attribute values are provided, including use of [Regular Expressions](#) and predefined enumerated value lists to constrain element and attribute values. DTD's do not allow use of regular expressions and only allow enumerated value lists for attributes.

For activities that intend to migrate towards XML Schemas, an excellent [free XML schema tutorial](#)<sup>iv</sup> is available from [Roger Costello](#)<sup>v</sup>; it provides both detailed presentations and hands-on labs. Additionally, a series of [XML Schema best practice papers](#)<sup>vi</sup> is available. These papers provide more XML Schema development technical detail than is provided here.

## 6.2.2. Recommended Schema Development Methodology

### Guidance

DON XML developers **SHOULD** adopt the practice of developing schemas based on information exchange requirements identified via business process modeling. The information modeling process and the XML schema creation process **SHOULD** be separate and distinct steps.

Business process models and corresponding document models describing information exchanged in the processes **MAY** use the Unified Modeling Language ([UML](#)) if appropriate. Specifically, the UN/CEFACT adopted Unified Modeling Methodology ([UMM](#)), based on UML, **MAY** be used for the process modeling. The DONXML WG expects to evaluate the UMM for applicability to DON data domains for possible official adoption at a later date.

Database modeling languages that are primarily oriented towards describing information via relational (keyed) structures **SHOULD NOT** be used for modeling of systems and information that will primarily use XML as the data exchange format.

Schema development **SHOULD** take place as a team effort with functional data experts, business experts, program managers, and IT specialists all involved. The DONXML WG also highly encourages collaboration between activities developing schemas within related information domains.

Conversely, schema development **SHOULD NOT** be solely the function of IT specialists. XML component names in general **SHOULD NOT** be taken directly from underlying relational database table and column

## Initial XML Developer's Guide - 29 October 2001

names, unless the elements within that database have been named and created in accordance with a DON or DOD standard that represents concurrence by an entire Community of Interest (COI).

### Explanation

The single most critical factor in creating logical, reusable schemas for information exchange in XML is the separation of the information modeling process from the schema creation process. Information should be modeled independently of creating a schema. This allows stakeholders to focus on creating logical, consistent representations of information, without getting distracted by the myriad of schema design options that have nothing to do with the information. Once an agreed to information model has been created, mapping rules from the model to a schema can be used or developed, which make schema creation straightforward. Just as this is the most important step, it is the most often neglected.

Typically, newly trained or inexperienced developer's begin creating schema's on an ad hoc basis, without the involvement of business functional experts and without a carefully crafted information model that lends itself to expressing hierarchical, object like relationships. Often, application developer's working without management and functional involvement and without an appropriate model are tempted to create XML quickly and easily from relational database table and column names. [XML components](#) produced in this fashion have very terse, abbreviated and generally unreadable names, which are often not reusable by other systems or concurred with by the community of users.

The result of the actions in the above paragraph is inevitably a poorly designed set of schemas with little reusability, extensibility, or readability; this translates into rework later at additional expense.

Because most uses of XML can be conceptualized as business processes in which communities of users share information; successful schema development should be based on analyzing, documenting, and reaching consensus on the *business processes*, the parcels of information (documents) exchanged in those processes, and the structure of a commonly understood vocabulary / [grammar](#) for creating the documents.

The focus of XML [schema](#) and [component](#) development should be on creating XML-languages that are understood by a community of stakeholders that engage in business processes together. In this context, the term *business process* is used in a larger scope than just business-to-business transactions (B2B) where products are bought and sold for money. Some examples:

- A supply activity wishes to make available, to its community, reference tables of code lists in an XML format. Here the process is consumer-to-application (C2A) / application-to-consumer (A2C) and application-to-application (A2A). A user (consumer) may request the table data via a web-browser (C2A); the activity receives the request and returns XML that is transformed to HTML (A2C). Also, an application may request and receive the same information in XML format via [SOAP](#) (A2A).
- A C4ISR application wishes to make air tasking order information, from messages, available on a publish-subscribe or broadcast basis to both operators and other C4ISR applications.
- A logistics activity wishes to store product data from an acquisition in a neutral format so that at some future point it can be [parsed](#) and read into any database for future processing by other activities needing it. In this case the process can be thought of as consumer-to-consumer (C2C), because the product data that is received by the acquiring consumer should be represented in an XML language that is understood by other consumers within the community.

Relational modeling languages, like IDEF1x, are appropriate for logical and physical enterprise data modeling of complex systems or data warehouses that will be implemented primarily by relational data bases. However, they are not appropriate for modeling hierarchical, object-like relationships expressed by XML. Relational modeling focuses the efforts of the modeling exercise on the efficient representation of data as a set of normalized entities; this simplifies the process of creating relational databases but complicates the process of understanding the information and it often hides or neglects critical object like aspects of the domain.

XML is an information sharing metalanguage that is inherently hierarchical, lending itself to be better represented via graphical modeling languages, which allow capture of object relationships vice key/key-

**Initial XML Developer's Guide - 29 October 2001**

reference relationships of normalized entities. The DONXML WG recommends that activities interested in capitalizing on XML as an information exchange medium take the time to learn the [UML](#). UML is rapidly becoming the de facto industry standard for system requirements analysis, business process and information modeling as well as software design. It provides a common language that business experts, managers and IT specialist can use throughout all phases of a system's implementation (requirements discovery, analysis, business rules and workflow documentation, software design, and deployment).

Many data-modeling languages have an object orientation, however, products supporting the direct creation of XML DTDs and/or Schema from UML are becoming available, and the [UN/CEFACT Electronic Business Transition Working Group](#)<sup>vii</sup> is undertaking to standardize a [UML to XML](#)<sup>viii</sup> mapping that will even further improve future tool support. By taking the time to create UML static structure models of information exchange requirements, schemas can be automatically generated and updated as standards and models evolve. This will ultimately drive down the cost of implementing XML based systems.

UML to XML tools are in their infancy. Due to lack of a standard, each tool does it differently at present. However, by taking the time to learn UML now, and beginning the process of creating information models in UML, DON activities will be well positioned to capitalize on future advancements.

**Examples**

A proposed procedure for schema development is presented in [Appendix E](#). It is non-[normative](#), and provided as an example only.

**6.2.3. Capturing Metadata****Guidance**

DON XML developers **SHOULD, within reason**, capture as much [metadata](#) as possible in a [schema](#).

The schema language chosen ([DTD's](#) or [XML Schema](#)) will impact the amount of metadata that can be expressed, and how well applications can access the metadata for processing.

- For DTD's, [XML comments](#) **MAY** be used to annotate the DTD with definitions and constraints, which the DTD syntax doesn't allow.
- Alternatively, for DTDs, fixed attributes **MAY** be used to capture the metadata.
- For XML Schema, metadata may be captured in a number of ways, as is discussed in the following sections. The four primary ways of capturing metadata are:
  - Domain value restrictions **SHOULD** be captured by the use of built-in Schema data types, the construction of custom data types, the assignment of enumerations to [XML component](#) values, the use of [regular expressions](#), and minimum / maximum value constraints.
  - Metadata regarding the structure and cardinality of components **SHOULD** be captured by expressing element order as either a (set of) choice(s), an ordered sequence, or as unordered. Additionally, the exact number of times an element can, or must, be repeated **MAY** be specified.
  - Logical relationships or relationships to existing data dictionaries and models (such as the [DDDS](#), ebXML [core components](#), or COE Reference Data Sets) may be expressed by the use of [types](#) or [Schema annotations](#).
  - An element's definition, sources of definitions or code lists, version information, and other metadata **MAY** be captured by the use of [Schema annotations](#).
- Developer's **MAY** consider the creation of a verbose *semantic schema* and a *compact schema* strictly for document validation purpose.
- Alternatively, schema documentation and annotations **MAY** be provided by creating a **schema guide** that is [URL](#) accessible and referenced in the header of the schema. Tools such as XML

## Initial XML Developer's Guide - 29 October 2001

Spy 4.x provide excellent documentation generation capabilities that can partially automate this process.

### Explanation

The [schema](#) is more than just a [document](#) structure validation tool. The XML Schema language, in particular, has a rich feature set for capturing extra metadata that can provide:

- Data element definitions through the use of annotations
- Detailed domain value constraints
- Logical data element pedigree through the use of annotations and types.

By capturing this metadata, the schema becomes an interoperability tool, because analysts can read it and understand what the various [XML components](#) mean and where they are derived from. Several sources of metadata exist that can be used to derive XML components, these include:

- The [COE XML Registry](#)<sup>ix</sup>.
- The initial set of ebXML core components (see [the ebXML Technical Reports](#)<sup>x</sup> on Core Components)
- The [DDDS](#)
- The [COE Data Emporium Reference Data Sets](#)<sup>xi</sup>.
- Various Military Standards ([MIL-STD-6040](#)<sup>xii</sup>, 6011, 6016, etc.)

With the exception of the COE XML Registry, the sources named do not provide readily reusable XML component names, however they do provide agreed to, reusable data element definitions.

A fully documented XML Schema may be quite verbose. Such “semantic” Schemas can provide critical insight to analysts desiring to understand and interoperate by making use of the information in the Schema. However, they contain much more information than is really necessary for document structure validation. A “compact” Schema that is equivalent to the “semantic” Schema may be quickly built for validation purposes. Having both a full “semantic” Schema and a “compact” schema may be appropriate for activities wishing to provide extensive Schema annotations, or underlying [type](#) relationships while having a smaller schema used strictly for validation.

A *schema guide* document that fully defines and explains each component in schema and the schemas logical structure is an alternative to creating a fully documented semantic schema.

### Example

[Appendix E](#) provides an example that combines several of the concepts discussed so far, including capturing definitions and relationships.

#### 6.2.3.1. Application Specific Metadata

##### Guidance

Application specific metadata (such as [SQL](#) statements or [API](#) calls) that is of interest only to a single application **SHALL NOT** be included in [instances](#) or [schemas](#).

##### Explanation

Including application specific metadata in an [instance](#) unnecessarily clutters the [document](#), increases bandwidth requirements, and is only useful to one application.

#### 6.2.3.2. Capturing Data Element Definitions



## Initial XML Developer's Guide - 29 October 2001

### Guidance

DON XML developers **MUST**, through [XML comments](#) or [XML Schema annotations](#), document [XML element](#) and [XML Schema type](#) definitions.

Definitions **SHOULD** be brief and when possible be taken from existing standard data element definitions, such as those provided by the [DDDS](#), [ebXML Core Components](#), [COE Reference Data Sets](#), or other Military Standards ([MIL-STD-6040](#), 6011, 6016, etc.)

Definitions **SHOULD** contain URL or other pointers to the definition's source, so that analysts can look up additional information.

Developers **MAY** extend the [XML Schema annotation](#) `<xsd:documentation>` tag by further marking up information provided with custom tags. No standards for this yet exist; however, the general guidelines of this document should be followed, and custom [metadata](#) tag names should follow the naming convention of the source data dictionary.

Developers **MAY** elect to publish schema documentation in a separate **schema guide**, however if this option is chosen, the schema must be [URL](#) accessible and referenced in the schema header.

### Explanation

Many activities in the DON are rapidly developing schemas as part of initiative such as TFWeb. Mandating that schema developers take the time to provide element and Schema type definitions will facilitate identifying commonalities and reusable components. Furthermore, it will start to enforce some rigor and thought in the creation of XML components, as business and technical experts come together to create definitions for components, and map their context specific elements back to applicable DON and DOD enterprise data standards.

[Section 6.4](#) provides guidance on use of XML elements vice [attributes](#). It is the DONXML WG's recommendation that attributes be minimized, and only used to provide supplementary metadata necessary to understand the business value of an XML element. By adopting this convention, and that of naming attributes in [camel case](#) according to [ISO 11179](#) conventions, attributes will be reasonably self-explanatory and therefore not require a definition in most cases.

### Example

[Appendix E](#) provides a consolidated example of capturing data element definitions in XML Schema.

Examples Section [6.1.2](#) also illustrates these concepts.

### 6.2.3.3. Enumerations and Capturing Code Lists

#### Guidance

DON XML schema developer **SHOULD** use [XML Schemas](#) to express enumeration constraints on [XML element](#) and [attribute](#) values, when such enumerated lists are of reasonable length and when code lists are considered stable (not likely to change frequently).

The decision to explicitly enumerate in a schema **SHOULD** be made by program managers based on the resulting size of the schema, bandwidth availability, and validation requirements.

Code lists, from which enumerations are taken, **SHOULD** be referenced by URI or other pointers so that analysts can lookup code values.

#### Explanation

The DOD and DON frequently represent data element values as codes rather than as free text. Codes are much easier for an application to understand and process because they are taken from a finite list of possible values, each with agreed upon semantics. Application developers create software to execute actions based on those code definitions and a specified set of business rules. XML can be used to exchange data that uses codes to abbreviate information, and the schema can be used to provide

**Initial XML Developer's Guide - 29 October 2001**

metadata about codes and their associated definitions (reference tables). Again, the way this is accomplished depends on the [schema](#) language chosen, with [XML Schemas](#) offering the most functionality. Capturing a reference to a list of valid codes and code values will greatly enhance implementations and allow future analysis to identify standard code reference tables. However, for code lists that historically change frequently, a URI pointer to the authoritative code list source is preferable.

**Example**

A [DTD](#) example of an element taken from the MIL-STD-6040 (USMTF) with an enumerated set of possible values and an [XML comment](#) referencing the source of the code definitions. Note, the only way to express an enumeration in a DTD is via an attribute. In this example, the '**casualtyCategoryCode**' [attribute](#) is better made an [XML element](#) (see [Section 6.4](#)). Use of the XML Schema language would have allowed expressing this enumeration as an element.

```
<!ELEMENT Casualty EMPTY>
<!ATTLIST Casualty casualtyCategoryCode (1 | 2 | 3 | 4 ) #REQUIRED>
<!-- casualtyCategoryCode
Definition: A CATEGORY DENOTING THE EFFECT OF A CASUALTY ON A UNIT'S
PRIMARY AND/OR SECONDARY MISSION AREAS.
Source: MIL-STD-6040 Baseline 2001 FFIRN 1207 FUDN 0001 -->
```

## 6.3. Document Annotations

**Guidance**

DON XML schema developers **MUST** provide carefully thought out comments within [schema](#) and [stylesheets](#), which provide basic information necessary to use and understand the document.

In general, Instances **SHOULD NOT** be documented; however, there may be situations where it is appropriate.

**Explanation**

Just as it is good programming practice to document application code using a coding standard, it is important that XML [documents](#) (instances, schemas, stylesheets) be well documented in a standard fashion. The follow paragraphs provide some recommended guidance.

The simplest way to express annotations is through the use of [XML Comments](#). Comments can be inserted anywhere in an XML [document](#) after the [XML Declaration](#).

[XML Schema annotations](#) provide a more flexible, extensible way to document Schemas as illustrated by many examples in this document.

### 6.3.1. Document Versioning

**Guidance**

Version information for [instances](#), [schemas](#), and [stylesheets](#) **MUST** be available via [document](#) annotations ([XML comments](#) or [Schema annotations](#)).

**Explanation**

Having a schema's version number available to developers in some form of annotation, both in the schema itself and in XML instances that conform to the schema, will assist in creating implementation that will maintain backward compatibility. Version information is also necessary for stylesheets in order to



**Initial XML Developer's Guide - 29 October 2001**

determine which version of a stylesheet correctly transforms an instance that conforms to a version of a schema.

### 6.3.1.1. Versioning DTD's

#### Guidance

DTD version information **SHOULD** be captured as an XML comment in the header of the DTD, and **MAY** be captured as a fixed [attribute](#) of the [root element](#).

#### Explanation

[DTD's](#) offer two means of documenting version number. The most straightforward is to put the DTD version number in the header XML comment. A second, preferred method available if developing in [XML Schema](#) is to declare a fixed schema version [attribute](#) to the XML [Root Element](#). This will make the version generally available to applications via an [API](#) call.

#### Example

```
<?xml version='1.0' encoding='UTF-8' ?>
<!ELEMENT root EMPTY>
<!ATTLIST root    schemaVersion CDATA    #FIXED '1.0' >
```

Providing version information in an XML comment in the header of a schema is discussed in [Section 6.3.2](#).

### 6.3.1.2. Versioning XML Schemas

#### Guidance

[XML Schemas](#) **SHOULD** include the version number in the header comments and **SHOULD** capture the version in an annotation to the [root element](#) of the [document](#).

#### Explanation

The schema header as discussed in [Section 6.3.2](#) provides a uniform method to capture a consistent body of information required for a schema. However, developers can make version information more easily available to applications through the use of the `<xsd:appInfo>` [tag](#) as shown in the example.

#### Example

Example of using [Schema annotations](#) to capture schema version information in an `<xsd:appInfo>` tag:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">
  <xsd:annotation>
    <xsd:appinfo>
      <Version>1.0</Version>
    </xsd:appinfo>
  </xsd:annotation>

```

```

</xsd:annotation>

<xsd:element name="root" />

</xsd:schema>

```

### 6.3.1.3. Versioning Stylesheets

#### Guidance

A [stylesheet](#) **MUST** contain references to the name and versions of the [schema](#) that describe [instances](#) upon which the stylesheet performs correctly.

#### Explanation

Tracking versions of stylesheets is very important because a new version of a stylesheet may or may not correctly transform an instance conforming to an old version of a schema. Explicitly asserting in a stylesheet which versions of a schema are supported will alleviate potential interoperability issues as implementations evolve.

#### Example

Example of using a custom '**schemaVersion**' attribute to express a series of schema versions, which a new XSLT stylesheet version will correctly transform. Note: the value of the '**schemaVersion**' attribute is a white space separated series of [Name Tokens](#).

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" schemaVersion="V1.0
  V1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
    indent="yes" />

  ...

</xsl:stylesheet>

```

### 6.3.2. Headers

#### Guidance

To promote interoperability, every [schema](#), [stylesheet](#), or [document](#) **MUST** contain some basic metadata.

The following metadata **SHOULD** be provided:

- Schema:
  - Schema Name
  - Schema Version
  - COE Namespace(s)
  - Navy Functional Data Area
  - URL to most current version

## Initial XML Developer's Guide - 29 October 2001

- For XML Schema, other Schemas imported or included to include COE Namespace, Schema file name, and URL.
  - For DTD, external entities referenced to include file name and URL
  - A description of the purpose of the schema
  - The name of the application or program of record that created and and/or manages the schema
  - The version of the application or program of record
  - A short description of the application interface that uses the description. A URL reference to a more detailed interface description may be provided
  - The name and versions of any associated stylesheets
  - Developer point of contact information to include activity, name and email
  - A change history log that includes change number, version, date and change description
- Stylesheets:
  - Stylesheet Name
  - Stylesheet Version
  - A list of schemas and XSL processors that the stylesheet have been tested against
  - The COE Namespace where the stylesheet is registered
  - Navy Functional Data Area of the application that makes use of the stylesheet
  - URL to most current version
  - Other stylesheets imported to include name and URL
  - A description of the purpose and function of the stylesheet
  - Application or program of record (with version) responsible for developing and maintaining the stylesheet
  - Developer point of contact information to include activity, name and email
  - A change history log that includes change number, version, date and change description
- Instances:
  - The name and URL of the schema that validates, and the stylesheet (if any) that correctly transforms it, if these are not specified already as part of the instance.

### Explanation

Other interested parties must be able to read a [document](#) and understand how to implement it or use information from it. Much of the information captured in a header XML comment can be better made available to applications through the use of fixed attributes or XML Schema annotations. However, having a consistent set of header information in a consistent location in an XML document will promote better configuration management and interoperability as methods for making this information available to applications are standardized. While examples are provided that show the above information captured in a single comment after the [XML declaration](#), this should not discourage innovative developers from providing the same information as Schema annotations (possible with custom markup inside a `<xsd:documentation>` tag. Some information may also be captured as fixed attributes if developing in DTDs, as illustrated by previous examples.

### Example

[Appendix F](#) provides non-[normative](#) examples of document headers.

## 6.4. Attributes vs. Elements

### Guidance

The number of [attributes](#) **SHOULD** be carefully considered and in general used sparingly.

Attributes, if used, **SHOULD** provide extra metadata required to better understand the business value of an element.

Attributes **MAY** be used to express code values while the content of the code (the definition) **MAY** be located as the element value.

Some additional guidelines are:

- Attribute values **SHOULD** be short, preferably numbers or conforming to the [XML Name Token](#) convention. Attributes with long string values **SHOULD NOT** be created.
- Attributes **SHOULD** only be used to describe information units that cannot or will not be further extended, or subdivided.
- Information specific to a single application or database **MUST NOT** be expressed as values of attributes (see [Section 5.2.3.1](#))
- Use attributes to provide metadata that describes the entire contents of an element. If the element has [children](#), any attributes should be generally applicable to all the children.

### Explanation

One of the key schema design decisions is whether to represent an information element as an [XML element](#) or attribute. Once an information element has been made an attribute, it cannot be extended further; for this reason and to promote better uniformity within the DON, the use of attributes is recommended only sparingly.

One of the key issues with attributes is that attribute values are easily accessed with less possible white space processing difficulty, as compared to element values, when attribute values are restricted to [XML Name Tokens](#) or numbers; this is because an XML [parser](#) will normalize<sup>2</sup> white space in attribute values. Also attribute values are not as easily available via [SAX](#); therefore processing an XML instance with a large number of attributes and attempting to access the information via SAX may be difficult.

---

<sup>2</sup> "Normalization" is accomplished by stripping all leading and trailing white space characters and reducing all white space between non-white space characters to a single blank space (#x20).

**Example**

Below the code KTS (for knots) provides extra [metadata](#) required to understand the 'business value' of the element – 600. It answers the question, "600 what?"

In the other examples, several ways of expressing coded values are illustrated.

```
<TargetVelocityMeasure measureUnitCode="KTS">600</
TargetVelocityMeasure >

or

<CasualtyCategoryCode value="1"> [TRAINING ACTIVITY ONLY]
EQUIPMENT CASUALTY EXISTS BUT WILL NOT IMPACT TRAINING
WITHIN 30 DAYS. </CasualtyCategoryCode>

or

<CasualtyCategoryCode value="1"/>
<CasualtyCategoryCode>
  <CodeContent>1</CasualtyCategoryCode>
  <CodeName>[TRAINING ACTIVITY ONLY] EQUIPMENT CASUALTY
  EXISTS BUT WILL NOT IMPACT TRAINING WITHIN 30 DAYS.
  </CodeName>
</CasualtyCategoryCode>
```

## 7. COE XML Registry

**Guidance**

Reference (a) **REQUIRES** all DON developers to reuse existing [tags](#) in the [COE XML Registry](#), if sufficient, or re-use commercial industry standard vocabularies if applicable, before developing their own.

It furthermore **REQUIRES** activities to register developed [XML Components](#) with the COE XML Registry.

Developers **MUST** familiarize themselves with this site and the associated [COE Namespaces](#)<sup>3</sup>. Each activity submitting a [registration package](#) to the registry is **REQUIRED** to do so to a specific COE Namespace via the [Namespace Manager](#).

**Explanation**

While this guidance provides many recommendations and examples of how to create more interoperable XML, the single biggest factors affecting interoperability are visibility and reuse. A draft DOD policy establishes the Defense Information Systems Agency (DISA) as the lead for the single DOD point of entry for XML registry and repository functions. Currently, DISA sponsors and has made available the [COE XML Registry](#). The intent of the COE Registry is to provide visibility into XML components that are being used throughout the DOD.

The DONXML WG is working with COE representatives to develop specific guidance for developers as to which Namespace they should register with. Until this is promulgated, activities should study the Namespace descriptions on the registry site, and contact the Namespace manager for what appears to

<sup>3</sup> A [COE Namespace](#) and an [XML Namespace](#) are not the same thing. Be sure and understand the difference.

## **Initial XML Developer's Guide - 29 October 2001**

be the most appropriate place for registration. If unable to locate an appropriate Namespace, register with the '*To Be Determined*' (TBD) Namespace.

Pending resolution, a single application should submit its registration package to a single [COE Namespace](#). In the case where an application's data crosses COE Namespace boundaries, request the [COE Namespace Manager](#) to provide guidance.

### **Example**

An example of a COE Registration package was obtained from the COE XML Registry and is available for [download](#) from the [NavyXML Quickplace](#) library.

## **8. Points of Contact**

### **DONXML WG Government Lead:**

Michael Jacobs, [Jacobs.Michael@hq.navy.mil](mailto:Jacobs.Michael@hq.navy.mil) , (703) 601-3594

### **DONXML TechTeam Lead and Editor:**

Brian Hopkins, [bhopkins@logicon.com](mailto:bhopkins@logicon.com), (858) 597-7293

## 9. Appendices

The following appendices are presented in draft form. They represent the understanding and opinion of the editor, and not the consensus of the DONXML WG. They are provided, as-is, and are to be considered non-[normative](#). The only exceptions are the portions of the ebXML Specifications and Technical Reports quoted in Appendix A.

### Appendix A – ebXML and the eBTWG

#### Description

ebXML was a 18-month international project sponsored jointly by [OASIS](#)<sup>xiii</sup> and [UN/CEFACT](#)<sup>xiv</sup> that ended in May, 2001 with the delivery of several specifications, technical reports and white papers available at [www.ebxml.org/specs](http://www.ebxml.org/specs). The ebXML deliverables defines an architecture with two distinct views. The Functional Service View (FSV) defines:

- Functional capabilities;
- *Business Service Interfaces*;
- *Protocols and Messaging Services*.

In other words, the FSV consists of specifications and standards that describe how an ebXML compliant system will physically operate to include interfaces, protocols, and registry/repository operations.

The Business Operational View (BOV) addresses:

- a) The semantics of business data in transactions and associated data interchanges
- b) The architecture for business transactions, including:
  - Operational conventions;
  - Agreements and arrangements;
  - Mutual obligations and requirements.

The BOV work focused on two areas. The first focus was on creating a methodology by which business processes can be modeled as orchestrated collaborations between business partners who exchange [payloads](#) of information (which [may](#) be [XML documents](#)). The [UMM](#) was chosen as the modeling methodology and a [BPSS](#) was created. Second, the BOV work focused on creating a methodology for creating reusable components – process components which can be used to build complex business process models, and information components which can be used to construct business documents as payloads of ebXML messages. Some of the [ebXML technical reports](#) discuss the concept of [core components](#) as universal, domain independent information entities defined in an XML-neutral syntax. This is significant because the ebXML authors intentionally did not address how components (core and domain specific) should be used to produce business [documents](#) (in XML). According to the ebXML architecture, ebXML components exist as registered objects within an ebXML registry/repository system; the work of defining production rules for creating XML payloads from registry entries was deferred. This decision has drawn sharp criticism from some, however it makes sense. The ebXML strategy was to first address how to represent information (semantics and context) independently of how it is syntactically expressed as an XML document; consequently the ebXML technical reports on core components adopt the [ISO 11179](#) naming convention for creation of **dictionary entries** for **information entities**. They do not specify how to create [XML component](#) names for schemas describing business documents containing payloads of information.

The ebXML deliverables provide a basis for future work required to make the vision of global interoperability a reality. OASIS and UN/CEFACT agreed to divide that work between them with OASIS assuming responsibility for the FSV aspects while UN/CEFACT took on the BOV portion. Since that time, UN/CEFACT has established the Electronic Business Transition Working Group ([eBTWG](#)<sup>xv</sup>),

## Initial XML Developer's Guide - 29 October 2001 Appendix A

*“...for the purpose of continuing the UN/CEFACT's role in pioneering the development of XML standards for electronic business. The group was formed to build on the success of the earlier ebXML Joint Initiative between UN/CEFACT and OASIS, which delivered its first set of specifications in May 2001.”*

One of the key deliverables of this group will be a final Core Component Specification that will combine and further refine the [ebXML Core Component Technical Reports](#)<sup>xvi</sup>.

The rest of the information presented in this appendix is taken from the deliverables of the ebXML project. These documents are works in progress. They may be useful in selecting data element and XML component names; however developers must and should expect the rules and specifications presented here to evolve rapidly.

### ebXML Naming Rules

Quoted<sup>4</sup> from [the ebXML Technical Architecture](#)<sup>xvii</sup>, Section 4.3 *Design Conventions for ebXML Specifications*:

“In order to enforce a consistent capitalization and naming convention across all ebXML specifications "Upper Camel Case" (*UCC*) and "Lower Camel Case" (*LCC*) Capitalization styles **SHALL** be used. *UCC* style capitalizes the first character of each word and compounds the name. *LCC* style capitalizes the first character of each word except the first word.

1) ebXML DTD, XML Schema and XML instance documents SHALL have the effect of producing ebXML XML instance documents such that:

- Element names SHALL be in *UCC* convention (example: `<UpperCamelCaseElement/>`).
- Attribute names SHALL be in *LCC* convention (example: `<UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>`)...

3) General rules for all names are:

- Acronyms SHOULD be avoided, but in cases where they are used, the capitalization SHALL remain (example: XMLSignature).
- Underscore ( `_` ), periods ( `.` ) and dashes ( `-` ) **MUST NOT** be used (don't use: `header.manifest`, `stock_quote_5`, `commercial-transaction`, use `HeaderManifest`, `stockQuote5`, `CommercialTransaction` instead)."

The following is component naming rules as quoted from the technical report, [Naming Convention for Core Components](#)<sup>xviii</sup> Section 5.2. They are based on [the ISO 11179](#) Part 6 draft specification. In reading these:

- Substitute "[XML Component](#)" for "Dictionary Entry".
- Caveats and comments are inserted in brackets.

---

<sup>4</sup> Copyright © ebXML 2001. All Rights Reserved.

“This document and translations of it MAY be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation MAY be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself MAY not be modified in any way, such as by removing the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other than English.”



## Initial XML Developer's Guide - 29 October 2001 Appendix A

- Since the publication of this report, the eBTWG has changed “representation type” to “representation term”:

**Rule 1:** The Dictionary Entry Name shall be unique and shall consist of Object Class, a Property Term, and Representation Type.

**Rule 2:** The Object Class represents the logical data grouping (in a logical data model) to which a data element belongs” ([ISO 11179](#)). The Object Class is the part of a core component’s Dictionary Entry Name that represents an activity or object in a context.

An Object Class may be individual or aggregated from core components. It may be named by using more than one word.

**Rule 3:** The Property Term shall represent the distinguishing characteristic of the business entity. The Property Term shall occur naturally in the definition.

**Rule 4:** The Representation Type shall describe the form of the set of valid values for an information element<sup>5</sup>. It shall be one of the terms specified in the “list of Representation Types” as included in this document.

Note: If the Representation Type of an entry is “code” there is often a need for an additional entry for its textual representation. The Object Class and Property Term of such entries shall be the same.

(Example : “**Car. Colour. Code**” and “**Car. Colour. Text**”). [In applying this convention to XML, via [XML Schema](#), the textual description of a code can expressed as an XML [Schema annotation](#), and therefore is not required in the [instance](#).]

**Rule 5:** A Dictionary Entry Name shall not contain consecutive redundant words. If the Property Term uses the same word as the Representation Type, this word shall be removed from the Property Term part of the Dictionary Entry Name.

For example: If the Object Class is “goods”, the Property Term is “delivery date”, and Representation Type is “date”, the Dictionary Entry Name is ‘Goods. Delivery. Date’.

In adoption of this rule the Property Term “Identification” could be omitted if the Representation Type is “Identifier”.

For example: The identifier of a party (“Party. Identification. Identifier”) will be truncated to “Party. Identifier”.

**Rule 6:** One and only one Property Term is normally present in a Dictionary Entry Name although there may be circumstances where no property term is included; e.g. Currency Code.

**Rule 7:** The Representation Type shall be present in a Dictionary Entry Name. It must not be truncated.

**Rule 8:** To identify an object or a person by its name the Representation Type “name” shall be used.

**Rule 9:** A Dictionary Entry Name and all its components shall be in singular form unless the concept itself is plural; e.g. goods.

**Rule 10:** An Object Class as well as a Property Term may be composed of one or more words.

**Rule 11:** The components of a Dictionary Entry Name shall be separated by dots followed by a space character. The words in multi-word Object Classes and multi-word Property Terms shall be separated by the space character. Every word shall start with a capital letter [In applying this to XML, use the upper and lower [camel case](#) convention, omit the periods and the spacing.]

---

<sup>5</sup> The term ‘information element’ is used generically in the same context as the term data element, and should not be confused with [XML Elements](#). An information element (or entity as ebXML refers to them) can be expressed as any of several XML components (XML Elements, attributes, or XML Schema types).

**Initial XML Developer's Guide - 29 October 2001 Appendix A**

**Rule 12:** Non-letter characters may only be used if required by language rules.

**Rule 13:** Abbreviations, acronyms and initials shall not be used as part of a Dictionary Entry Name, except where they are used within business terms like real words; e.g. EAN.UCC global location number, DUNS number [see section [5.1.2 Usage of Acronyms and Abbreviations](#)]

**Rule 14:** All accepted acronyms and abbreviations shall be included in an ebXML glossary [read, "...included in the element definition in the [schema annotation](#), see section [5.1.2](#)]."

## Representation Terms

The following extract is provided from a 12 October 2001 draft of the eBTWG core component specification. It is provided for information only:

"Table 6-3 Representation Terms

Representation Term	Definition	Links to Core Component Type
<b>Amount</b>	A number of monetary units specified in a currency where the unit of currency is explicit or implied.	Amount. Type
<b>Code</b>	A character string (letters, figures or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of an attribute. Codes usually are maintained in code lists per attribute type (e.g. colour).	Code. Type
<b>Date</b>	A day within a particular calendar year (ISO 8601).	Date Time. Type
<b>Date Time</b>	A particular point in the progression of time (ISO 8601).	Date Time. Type
<b>Graphic</b>	A diagram, graph, mathematical curves, or similar representation	Graphic. Type
<b>Identifier</b>	A character string used to identify and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme.  [Note: Type shall not be used when a person or an object is identified by its name. In this case the Representation Term "Name" shall be used.]	Identifier. Type
<b>Indicator</b>	A list of two, and only two, values that indicate a condition such as on/off; true/false etc. (synonym: "Boolean").	Indicator. Type
<b>Measure</b>	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE Rec. 20.	Measure. Type

## Initial XML Developer's Guide - 29 October 2001 Appendix A

Representation Term	Definition	Links to Core Component Type
Name	A word or phrase that constitutes the distinctive designation of a person, place, thing or concept.	Text. Type
Percent	A rate expressed in hundredths between two values that have the same unit of measure.	Numeric. Type
Picture	A visual representation of a person, object, or scene	Picture. Type
Quantity	A number of non-monetary units. It is associated with the indication of objects. Quantities need to be specified with a unit of quantity.	Quantity. Type
Rate	A quantity or amount measured with respect to another measured quantity or amount, or a fixed or appropriate charge, cost or value e.g. US Dollars per hour, US Dollars per EURO, kilometre per litre, etc.	Numeric. Type
Text	A character string generally in the form of words of a language.	Text. Type
Time	The time within a (not specified) day (ISO 8601).	Date Time. Type
Value	Numeric information that is assigned or is determined by calculation, counting or sequencing. It does not require a unit of quantity or a unit of measure	Numeric. Type

The following representation terms apply to aggregate Core Components or Core Component types.

**Table 6-4 Other Representation Terms**

Representation Term	Definition	Links to Core Component Type
Details	The expression of the aggregation of Core Components to indicate higher levelled information entities	Not Applicable
Type	The expression of the aggregation of Core Components to indicate the aggregation of lower levelled information entities to become Core Component Types. All Core Component Types shall use this Representation Term	Not Applicable
Content	The actual content of an information entity. Content is the first information entity in a Core Component Type	Used with the content components of Core Component Types

The ebXML core components technical reports require that name of “aggregate information entities” use the special representation type, ‘details’. DON XML developers may omit the term ‘details’ from the end of [tag](#) names when XML element names are generated from the ISO 11179 name. For example, the ISO 11179 data element name **‘Address. Details’** would be represented in XML as **<Address>**.

**Initial XML Developer's Guide - 29 October 2001 Appendix A**

The Representation Terms provided by ISO 11179 may not be adequate for a number of engineering, scientific and operational concepts. In these cases, use of other term names temporarily, such as until the list of types is expanded, **SHOULD** be considered; however do this with caution.

## Appendix B – Schema Development

### Possible Schema Development Procedure Summary

The following is presented as a possible procedure for developing schema. It does not represent the consensus of the DON XML WG; rather it is presented for your consideration and [feedback](#). It is purely developmental; all or none of it may be found useful.

#### STEPS

In creating XML components according to these conventions, try the following :

- Step 1. Analyze the business processes in which your application will exchange, use or store information. Understand who the consumers (both human and machine) of the information your application provides are. The DONXML WG recommends the use of the [UMM](#) and [UML](#) for this process, however any model that provides a basic understanding of how information will be exchanged across system boundaries (application to application, application to human, or human to application) can provide a basis for development as more rigorous modeling techniques, such as the UMM, are learned. The business process modeling should identify and name actors (persons, organizations, or systems) that participate in the process. The roles that each actor plays should also be identified and named. It is important to separate the name of the actor from the name of the role because often the same actor will participate in multiple roles within a process.
- Step 2. Based on the information exchange requirements identified in step 1, spend the time to model the data in each [document](#) that will be exchanged within the processes defined in step 1. DONXML strongly recommends using the Unified Modeling Language ([UML](#)) to conduct the modeling. Several efforts are underway to create production rules by which UML models can be directly used to generate XML documents. An excellent online resource is [xmlmodeling.com](#).
- Step 3. Look for previously developed XML components that can be reused, either in the [COE XML Registry](#) or schema developed by commercial consortia ([Appendix D](#) provides references).
- Step 4. Create the ebXML/[ISO 11179](#) compliant name and definition for each element identified in step 2 that will be used in an information exchange scenario.
- Step 5. Identify extra metadata required to understand the business value of each element. This extra metadata may be expressed in either the [schema](#) or the [instance](#) as [attributes](#) ([section 5.4 Attributes versus Elements](#) provides detailed guidance).
- Step 6. Analyze the information element. Ensure you have identified [specific](#) physical elements for each data item that will appear in the XML [instance](#). This process will help the team identify underlying logical elements or [generic](#) physical elements that can be reused by declaring them as [XML Schema Types](#) or as [abstract](#) elements. This analysis should supplement the model you defined in step 2, and may require that you iterate through step 2 again. The [UML](#) static structure artifact is extremely useful here. Last, determine relationships between elements defined here and existing data models and definitions (such as the [ebXML core components](#), the [DDDS](#), the [COE XML Registry](#) and [Data Emporium](#)).
- Step 7. Identify any common [business terms](#) that are associated with the information elements defined in step 2. If any are identified, one or more of these will be used as the actual [XML element](#) names.
- Step 8. Create the [schema](#) <sup>6</sup>.

---

<sup>6</sup> Up until now, we have not considered how we will express the information in XML. It is a good XML engineering practice to go through the process of defining and modeling information before the additional

## Initial XML Developer's Guide - 29 October 2001 Appendix B

- a. If creating schema as a DTDs, your choices are to make the model elements just defined an XML [element](#) or an [attribute](#)
- b. If employing the [XML Schema](#) language, you have some extra choices in deciding how to express an model element. Model elements can be expressed:
  - As [types](#), which may be declared [abstract](#).
  - As [abstract XML elements](#).
  - As (non-abstract) XML elements or [attributes](#).

One strategy for creating XML Schemas is as follows:

- Create an underlying set of simple and complex XML Schema types describing base data types, reusable logical and generic physical elements.
  - Declare every model element that will appear in the XML instance as type that derives from the types declared previously.
  - Create XML Schema types and attributes using the same name as the [ISO 11179](#) named model elements
  - Create XML elements names according to business terms, actor and role names. For instance **<TransmitterUnit>** is a tag name consisting of a role name and an actor name. **<AcousticFrequency>** is a business term for '**Acoustic Signal. Frequency. Measure**'. When no business term, or actor/role exists, consider creating element names that consist of an optional context term plus the ISO 11179 Object Class (plus property term if appropriate) plus representation term. For example **<DODMaterialItemIdentifier>**, where the context term is "DOD" indicating that the element is specific to the Department of Defense.
  - For business terms with commonly used synonyms, such as NSN for National Stock Number, create a substitution group for the additional synonyms.
- c. Build the schema from the bottom-up and top-down.

*Step 9.* Register any newly created XML elements with the COE XML Registry.

---

complications and design alternatives of XML are addressed. Trying to do both information modeling and XML design at the same time is confusing, and often, critical aspects of one or the other are missed.

## Appendix C - Tools and References

### Tools

Tools for developing and employing XML in applications are flooding the market. However, most if not all of these tools are in early stages of development. In future revisions to this publication, recommendations will be provided as to tools that have either been used, evaluated or are known by reputation. Pros and cons of each will be presented in the case where they are known. Application developer's that have used a particular tool may request that it be included in this list, provided it meets at least two of the following criteria:

- It is relatively mature or produced by an established vendor (such as IBM or Microsoft). A beta tool from Microsoft, or from IBM Alphaworks may be included, however a beta tool from CrazyXMLTools.com should not.
- It is a leader in a developing area, such as X2X's XLink processor. While still immature, it is currently one of the leaders in XLink processing software.
- It has been used by a Navy activity and found to be useful and relatively free of bugs, or the bugs are well documented.
- It has been evaluated by a neutral third part (such as Forrester or the Gartner Group, or an established periodical) with favorable results.

Submit proposed tools to the [editor](#) using the format of the following table:

<i>Name &amp; Link</i>	<i>Description</i>	<i>Pro's</i>	<i>Con's</i>
<b><i>XML, XSL and Schema Development</i></b>			
<b><i>XML Parsers and XSL Processors</i></b>			
<b><i>Databases</i></b>			
<b><i>"Servers"</i></b>			
<b><i>Miscellaneous</i></b>			

A more complete list of available XML software is maintained at [www.xmlsoftware.com](http://www.xmlsoftware.com).

### Publications

The following publications have been reviewed by the editor and found to be good reference material. The table presents several levels of reader and recommends appropriate reading for each.

<i><b>Audience</b></i>	<i><b>Title</b></i>	<i><b>ISBN</b></i>	<i><b>Author(s)</b></i>	<i><b>Date</b></i>
<b>Management /Business</b>	XML: A Manger's Guide	0-201-43335-4	Dick	2000
	ebXML: The New Global Standard for Doing Business on the Internet	0-735-71117-8	Kotok & Weber	2001
<b>Business / Technical</b>	XML in a Nutshell : A Desktop Quick Reference (Nutshell Handbook)	0-596-00058-8	Harold & Means	2001
	Metadata Solutions: Using Metamodels, Repositories, XML, and Enterprise Portals to Generate Information on Demand	0-201-71976-2	Tannenbaum	2001
	Modeling XML Applications with UML: Practical e-Business Applications	0-201-70915-5	Carlson	2001
<b>Technical</b>	The Wrox Professional <a href="#">XML Series</a>		<a href="#">Wrox</a>	
	Building B2B Applications with XML: A Resource Guide	0-471-40401-2	Fitzgerald	2001
	Java & XML, 2nd Edition: Solutions to Real-World Problems	0-596-00197-5	McLaughlin	2001
	SOAP: Cross Platform Internet Development Using XML	0-130-90763-4	Seely & Sharkey	2001
	Inside XSLT	0-735-71136-4	Holzner	2001
	XML Schema Development: An Object-Oriented Approach	0-672-32059-2	Brauer	2001

## Internet

BizTalk <http://www.biztalk.org/home/default.asp>

COE XML Registry: <http://diides.ncr.disa.mil/xmlreg/user/index.cfm>

ebXML <http://www.ebxml.org>

eBTWG <http://www.ebtwg.org/>

OASIS <http://www.oasis-open.org/>

Open Applications Group <http://www.openapplications.org/>

The Object Management Group [www.omg.org](http://www.omg.org)

RosettaNet <http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial>



DONXML TechTeam

**Initial XML Developer's Guide - 29 October 2001 Appendix C**

Schema.net <http://www.schema.net>

W3C <http://www.w3.org>

XML.com <http://www.xml.com/>

The XML Cover Pages <http://www.oasis-open.org/cover/sgml-xml.html>

XML Software.com <http://www.xmlsoftware.com/>

## Appendix D – W3C XML Recommendations

Appendix deleted. A current list may be found at the [W3C Technical Reports](#)<sup>xix</sup> page.

## Appendix E – Combined XML Schema Example

The following XML Schema is a combined example illustrating some of the guidance and concepts discussed in this document. The example is non-[normative](#), and does not represent the consensus of the DONXML TechTeam. It is provided for information only.

In this example, a [tag](#) from the [COE XML Registry](#), `<ACOUST_SIGNA_FREQ>` is reused, but the principles of [ISO 11179](#) and [camel case](#) are applied using the functionality of the [XML Schema language](#) to maintain interoperability.

The COE XML Registry defines a tag `<ACOUST_SIGNA_FREQ>` in the Tracks & Reports Namespace. An instance might look like this:

```
<ACOUST_SIGNA_FREQ>12.100</ACOUST_SIGNA_FREQ>
```

**Definition:** ACOUSTIC SIGNATURE FREQ. THE FREQUENCY OF AN EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH HERTZ.

Maximum Length: 10

You can view this tag definition at [http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir\\_id=8358](http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358).

A possible XML Schema for this element:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:complexType name="MeasureType">
    <xs:annotation>
      <xs:documentation source="http://www.ebxml.org/specs/ccDICT.pdf">
        <ebXML>
          <CoreComponent UID="core000152">Text. Type</CoreComponent>
        </ebXML>
      </xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="measureUnitCode" type="xs:string"
          use="optional" default="HZ" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="AcousticSignalFrequencyMeasure">
    <xs:annotation>
```

```

- <xs:documentation
  source="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358
">
- <COEXMLRegistry>
  <Namespace prefix="TAR">Tracks and Reports</Namespace>
  <TagName>ACOUST_SIGNA_FREQ</TagName>
  <Definition>acoustic SIGNATURE_FREQ. THE FREQUENCY OF AN
    EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH
    HERTZ.</Definition>
  <RegistryID>8358</RegistryID>
</COEXMLRegistry>
</xs:documentation>
- <xs:documentation
  source="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358
">
- <COEXMLRegistry>
  <Namespace prefix="TAR">Tracks and Reports</Namespace>
  <TagName>ACOUST_SIGNA_FREQ</TagName>
  <Definition>acoustic SIGNATURE_FREQ. THE FREQUENCY OF AN
    EMITTED ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH
    HERTZ.</Definition>
  <RegistryID>8358</RegistryID>
</COEXMLRegistry>
</xs:documentation>
</xs:annotation>
- <xs:simpleContent>
- <xs:restriction base="MeasureType">
  <xs:totalDigits value="10" />
  <xs:fractionDigits value="3" />
  <xs:pattern value="\d*\.\d{3}" />
  <xs:attribute name="measureUnitCode" fixed="HZ" />
</xs:restriction>
</xs:simpleContent>
</xs:complexType>
<xs:element name="ACOUST_SIGNA_FREQ"
  type="AcousticSignalFrequencyMeasure" />

```

```

<xs:element name="AcousticFrequency"
  type="AcousticSignalFrequencyMeasure"
  substitutionGroup="ACOUST_SIGNA_FREQ">
  <xs:annotation>
    <xs:documentation>Business Term </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:schema>

```

This Schema has 2 elements and two types declared:

Elements	Complex types
<b>ACOUST_SIGNA_FREQ</b>	<b>AcousticSignalFrequencyMeasure</b>
<b>AcousticFrequency</b>	<b>MeasureType</b>

The highest-level element:

element **ACOUST\_SIGNA\_FREQ**

diagram	<pre>graph BT; AcousticFrequency --&gt; ACOUST_SIGNA_FREQ; style ACOUST_SIGNA_FREQ fill:#fff,stroke:#333,stroke-width:1px; style AcousticFrequency fill:#fff,stroke:#333,stroke-width:1px;</pre> <p>Business Term</p>										
type	AcousticSignalFrequencyMeasure										
facets	<table><tr><td>totalDigits</td><td>10</td></tr><tr><td>fractionDigits</td><td>3</td></tr><tr><td>pattern</td><td>\d*.\d{3}</td></tr></table>					totalDigits	10	fractionDigits	3	pattern	\d*.\d{3}
totalDigits	10										
fractionDigits	3										
pattern	\d*.\d{3}										
attributes	Name	Type	Use	Default	Fixed						
	measureUnitCode				HZ						
source	<pre>&lt;xs:element name="ACOUST_SIGNA_FREQ" type="AcousticSignalFrequencyMeasure"/&gt;</pre>										

## Initial XML Developer's Guide - 29 October 2001 Appendix E

This element name is reused from the COE XML Registry. Also, note that the “facets” of the element specify the domain restrictions. The regular expression pattern “\d\*\d{3}” translates to, “Any number of digits followed by a period followed by 3 digits. We are relying on the total digits facet to constrain the length. The fraction digits facet specifies that there can be at most 3 digits past the decimal point, however the regular expression pattern requires that there be exactly three digits.

The element has one fixed attribute, measureUnitCode = “HZ”.

As indicated by its diagram, the above example has a business term:

element **AcousticFrequency**

diagram	<div><div>AcousticFrequency</div><div>Business Term</div></div>				
type	AcousticSignalFrequencyMeasure				
facets	totalDigits 10 fractionDigits 3 pattern \d*.\d{3}				
attributes	Name	Type	Use	Default	Fixed
	measureUnitCode				HZ
annotation	documentation	Business Term			
source	<xs:element name="AcousticFrequency" type="AcousticSignalFrequencyMeasure" substitutionGroup="ACCOUST_SIGNA_FREQ"> <xs:annotation> <xs:documentation>Business Term</xs:documentation> </xs:annotation> </xs:element>				

This business term is declared substitutable for first element because it is in the first element's [substitution group](#). It also has the same fixed attribute.

Notice that both this element and the first derive from the same type. You can determine this from the type reference in the above element declarations and tables or by looking at the “used by” row below.

complexType **AcousticSignalFrequencyMeasure**

diagram	<div> <div>AcousticSignalFrequencyMeasu...</div> <pre> &lt;COEXMLRegistry&gt;   □ &lt;Namespace prefix="TAR"&gt;Tracks and     Reports&lt;/Namespace&gt;   □   &lt;TagName&gt;ACOUST_SIGNAL_FREQ&lt;/Tag     Name&gt;   □ &lt;Definition&gt;acoustic SIGNATURE FREQ.     THE FREQUENCY OF AN EMITTED     ACOUSTIC SIGNAL TO THE NEAREST     ONE THOUSANDTH HERTZ.&lt;/Definition&gt;   □ &lt;RegistryID&gt;8358&lt;/RegistryID&gt; &lt;/COEXMLRegistry&gt; </pre> </div>				
type	restriction of MeasureType				
used by	elements <b>ACCOUST_SIGNAL_FREQ</b> <b>AcousticFrequency</b>				
facets	totalDigits <b>10</b> fractionDigits <b>3</b> pattern <b>\d*\d{3}</b>				
attributes	<b>Name</b>	<b>Type</b>	<b>Use</b>	<b>Default</b>	<b>Fixed</b>
	<b>measureUnitCode</b>	<b>xs:string</b>	<b>optional</b>		<b>HZ</b>
annotation	documentation <b>&lt;COEXMLRegistry&gt;</b> <b>    &lt;Namespace prefix="TAR"&gt;Tracks and</b> <b>    Reports&lt;/Namespace&gt;</b> <b>    &lt;TagName&gt;ACOUST_SIGNAL_FREQ&lt;/TagName&gt;</b> <b>    &lt;Definition&gt;acoustic SIGNATURE FREQ. THE</b> <b>    FREQUENCY OF AN EMITTED ACOUSTIC SIGNAL</b> <b>    TO THE NEAREST ONE THOUSANDTH HERTZ.</b> <b>    &lt;/Definition&gt;</b> <b>    &lt;RegistryID&gt;8358&lt;/RegistryID&gt;</b> <b>&lt;/COEXMLRegistry&gt;</b>				
source	<b>&lt;xs:complexType name="AcousticSignalFrequencyMeasure"&gt;</b> <b>  &lt;xs:annotation&gt;</b> <b>    &lt;xs:documentation</b>				


	<pre> <b>source</b>="http://diides.ncr.disa.mil/xmlreg/user/detail.cfm?ir_id=8358"&gt;   &lt;COEXMLRegistry&gt;     &lt;Namespace <b>prefix</b>="TAR"&gt;Tracks and Reports&lt;/Namespace&gt;     &lt;TagName&gt;ACOUST_SIGNA_FREQ&lt;/TagName&gt;     &lt;Definition&gt;acoustic SIGNATURE FREQ. THE FREQUENCY OF AN EMITTED     ACOUSTIC SIGNAL TO THE NEAREST ONE THOUSANDTH HERTZ.&lt;/Definition&gt;     &lt;RegistryID&gt;8358&lt;/RegistryID&gt;   &lt;/COEXMLRegistry&gt; &lt;/xs:documentation&gt; &lt;/xs:annotation&gt; &lt;xs:simpleContent&gt;   &lt;xs:restriction <b>base</b>="MeasureType"&gt;     &lt;xs:totalDigits <b>value</b>="10"/&gt;     &lt;xs:fractionDigits <b>value</b>="3"/&gt;     &lt;xs:pattern <b>value</b>="\d*\d{3}"/&gt;     &lt;xs:attribute <b>name</b>="measureUnitCode" <b>fixed</b>="HZ"/&gt;   &lt;/xs:restriction&gt; &lt;/xs:simpleContent&gt; &lt;/xs:complexType&gt; </pre>
--	---

It is in this [Schema type](#) that most of the declarations are made, and accordingly the most reuse is achieved. Note that the type name is the ISO 11179 name, and the documentation is taken from the [COE XML Registry](#). Also, the domain constraints that each of the two elements have are inherited from their definition here.

This has the '**measureUnitCode**' attribute; it is here that that attribute is declared as fixed. The two elements inherit the fixed attribute.

Note this type is derived from another type "by restriction" (see the 'type' row of the table below). The "by restriction" phrase means that this type is a more restrictive subset of a more generic type:

complexType **MeasureType**

diagram	 <pre> &lt;ebXML&gt;   □ &lt;CoreComponent   UID="core000152"&gt;Text,   Type&lt;/CoreComponent&gt; &lt;/ebXML&gt; </pre>
type	extension of xs:decimal
used by	complexType AcousticSignalFrequen



	cyMeasure				
attributes	Name	Type	Use	Default	Fixed
	measureUnitCode	xs:string	optional	HZ	
annotation	documentation <ebXML> 				

This type is created from the [ebXML core component](http://www.ebxml.org/specs/ccDICT.pdf), '*Measure. Type*' as indicated by its documentation elements. The UID attribute of the <CoreComponent> tag provides a reference back to the file pointed to by the documentation source attribute which is a URL, "<http://www.ebxml.org/specs/ccDICT.pdf>". It is an extension of the XML Schema base type 'decimal'. The extension is the addition of a 'measureUnitCode' attribute that is optional, with a default value of "HZ". Remember we changed it to a fixed attribute when we reused this type in the '*AcousticFrequencyMeasure*' schema type declared previously.

Note the use of customized tags to further markup information inside the <xs:documentation> tag.

**Initial XML Developer's Guide - 29 October 2001 Appendix E**

Some examples of XML instance fragments this document will validate:

```
<ACOUST_SIGNAL_FREQ>100.000</ACOUST_SIGNAL_FREQ>
or
<ACOUST_SIGNAL_FREQ
measureUnitCode="HZ">100.000</ACOUST_SIGNAL_FREQ>
or
<AcousticFrequency measureUnitCode="HZ">100.000</ AcousticFrequency >
```

## Appendix F – Sample XML Document Headers

### Sample Schema Header

```
<?xml version="1.0" encoding="UTF-8">
<!-- Schema/DTD Header *****
Schema Name: SPAWARVPO$2-1_FolderData$1-1.xsd
Schema Version: 1.1
COE Namespace(s): TBD
Navy Functional Data Area: Administration
Current version available at (URL): https://www.spawar.navy.mil/vpo/schemas/
Other Schemas Imported (XML Schema only):
**** Namespace Prefix: PER "http://diides.ncr.disa.mil/xmlreg/user/namespace_list.cfm"
**** Schema File Name: BUPERSBUPERSOnLine$3-0_Document$2-2.xsd
**** Available at URL: www.bupers.navy.mil/bupersOnLine/schemas/
Other Schemas Included (XML Schema only): None
External DTDs Referenced (DTD only): n/a
**** Name: n/a
**** Available at (URL): n/a
Description: Provides information regarding the content of VPO folders such as content file names, file
sizes, file owner, file status, and file access information.
Application: Virtual Program Office
Application Version: 2.1
Application Interface:
XML data is available from the VPO application via HTTP at
https://www.spawar.navy.mil/vpo/GetFolderInfo.asp. Input queries via HTTP GET with query string
format, "...?dir=directoryName". A complete interface description document is available at
https://www.spawar.navy.mil/vpo/interfaces/GetFolderInfo.txt
Associated Stylesheet:
**** Name: SPAWARVPO$2-1_ViewFolderContents$1-0.xsl
**** Available at (URL): https://www.spawar.navy.mil/vpo/stylesheets/
Developed by (Gov't Activity): SPAWAR 08
Point of Contact Name: Joe Smith
Point of Contact Email: jsmith@spawar.navy.mil
Change History:
CHANGE # Version      DATE      DESCRIPTION OF CHANGE
      0         1.0         15 Sep 2001      Initial release
```

1            1.1            30 Sep 2001            Updated to include file size information

\*\*\*\*\*

-->

This is a generic header that is provided in text-only, non-XML format. It can be used for either a DTD or XML Schema. A possibly more useful approach would be to markup header information using XML. The tags could be encapsulated by XML comment markup (<!-- ... --> or in the case of XML Schemas, included as an annotation following the XML Schema declaration. Marking up header information could be very useful; for instance a large number of schemas could be automatically analyzed to determine which COE Namespaces and Functional Data areas they fell into. This would be a time consuming manual process otherwise. The DONXML WG may work to standardize the tags and procedures for providing header information in XML markup. Until then, it is important to get the information somewhere in the document. Activities wishing to experiment with different strategies and techniques for providing header data are encouraged to do so and report there findings to the DONXML WG. Consider the above example the minimum information we think will be required; your input is encouraged.

### Notes on header fields:

Header Item	Description
<i>Schema Name:</i>	The standard name of the <a href="#">schema</a> file. See <a href="#">Document Naming Convention</a>
<i>Schema Version:</i>	The version of the schema. Adopt a major and minor version number separated by a '.'
<i>Tested With:</i>	List the name and version number of the <a href="#">XML processor</a> (s) that have been are tested known to corectly validate this schema.
<i>COE Namespace(s):</i>	Identify the <a href="#">COE Namespace</a> that the elements from this schema are registered in by specifying the <a href="#">COE XML Namespace Prefix</a> from the <a href="#">COE XML Registry</a> . You can specify mulple Namespaces for XML Schemas that use tags from mulitple COE Namespaces. This is only possible through the use of <a href="#">XML Schemas</a> because DTD's do not support <a href="#">XML Namespace</a> prefixing.
<i>Functional Data Area:</i>	Indicate which Navy Functional Data Area the application that uses this schema belongs to. Refer to the DMI Instruction (SECNAVINST 5000.36) and implementation guidance for a list.
<i>Current version available at (URL):</i>	If this schema is <a href="#">URL</a> accessable, put the address here. It is highly recommended that all <a href="#">schemas</a> be available on-line to assist other activities desiring to interoperate.
<i>Other Schemas Imported (XML Schema only):</i>	The <a href="#">XML Schema</a> language allows the reuse of existing XML Schema so that schemas can be modularized. The first way of doing this is via the XML Schema Import syntax.
<i>The next three fields are repeatable</i>	
<i>**** Namespace Prefix and URL:</i>	The XML Schema Import syntax is used when desiring to reuse a schema whose elements belong to a different <a href="#">XML Namespace</a> than the elements into which the import is being conducted on. Specify here the
<i>**** Schema File Name:</i>	The standard name of the imported schema file. See <a href="#">Document Naming Convention</a>
<i>*** Available at (URL):</i>	If this schema is <a href="#">URL</a> accessable, put the address here. It is highly recommended that all <a href="#">schemas</a> be available on-line to assist other activities desiring to interoperate.
<i>Other Schemas Included (XML</i>	The second way <a href="#">XML Schemas</a> allow reuse of other schemas is through the

## Initial XML Developer's Guide - 29 October 2001 Appendix F

<i>Schema only:</i>	XML Schema Include syntax. Includes can be used when the elements in the included schema belong to the same <u>XML Namespace</u> as the schema into which the include is occurring. A schema may both include and import.
<i>The next two fields are repeatable</i>	
**** <i>Schema File Name:</i>	The standard name of the imported schema file, see <u>Document Naming Convention</u>
*** <i>Available at (URL):</i>	If the schema file to be imported is <u>URL</u> accessible, put its address here. It is highly recommended that all <u>schemas</u> be available on-line to assist other activities desiring to interoperate.
<i>External DTDs Referenced (DTD only):</i>	Information regarding any <u>External Parameter Entity</u> references are made to an external DTD. This approximates the modular design capability available in XML Schema.
<i>The next two fields are repeatable</i>	
**** <i>Name:</i>	The standard name of the DTD file, see <u>Document Naming Convention</u>
**** <i>Available at (URL):</i>	If this schema DTD is <u>URL</u> accessible, put its address here. It is highly recommended that all <u>schema DTDs</u> be available on-line to assist other activities desiring to interoperate.
<i>Description:</i>	Plain text description of the type of information described by the schema.
<i>Application:</i>	The name of the application which produces XML <u>documents</u> that <u>validate</u> to this <u>schema</u> .
<i>Application Version:</i>	The version (major.minor) of the application that produces this schema.
<i>Application Interface:</i>	A plain text descriptive summary of how other applications interface with this application. For example, via HTTP, using query parameters passed via HTTP POST or GET. Examples of query name/value pairs may be provided. If SOAP is used, should provide a brief description of the method calls and parameters. A good XML engineering practice is to completely document your application interface; if you have done so, reference that documentation here. Making the interface specification available via a (secure) URL will assist other developers in interoperating.
<i>Associated Stylesheet:</i>	If a <u>stylesheet</u> is available to <u>render instances</u> that <u>validate</u> to this <u>schema</u> , provide information here.
**** <i>Name:</i>	The standard name of the stylesheet file, see <u>Document Naming Convention</u>
**** <i>Available at (URL)</i>	If the <u>stylesheet</u> is <u>URL</u> accessible, put the its address here. It is highly recommended that all stylesheets be available on-line to assist other activities desiring to interoperate.
<i>Developed by (Gov't Activity):</i>	Government Activity and Office code.
<i>Point of Contact Name: Joe Smith</i>	Name of person to contact with questionions regarding the schema.
<i>Change History:</i>	The following fields provide an audit trail of changes.
<i>CHANGE #</i>	Keep a sequentially numbered list of changes.
<i>Version</i>	You should also assign Major and minor version numbers.
<i>DATE</i>	Date implemented
<i>DESCRIPTION OF CHANGE</i>	Plain text description.

## Sample Stylesheet Header

This sample stylesheet header is similar to the schema header with the addition of information regarding which version of a schema the stylesheet is written from, and the removal of non-applicable items.

```
<?xml version="1.0">
<!-- Stylesheet Header *****
Stylesheet Name: SPAWARVPO$2-1_ViewFolderData$1-1.xsl
Stylesheet Version: 1.1
Tested to:
**** Schema Name: SPAWARVPO$2-1_FolderData$1-1.xsd
**** Schema Version: 1.1
**** XSL Processors: MSXML 3.0, XALAN 1.2.2
COE Namespace: TBD
Navy Functional Data Area: Administration
Current version available at (URL): https://www.spawar.navy.mil/vpo/stylesheet/
Other Stylesheets Imported:
**** File Name: BUPERSBUPERSONLine$3_Document$2-2.xsl
**** Available at URL: www.bupers.navy.mil/bupersOnLine/stylesheet/
Description: XSLT compliant stylesheet renders folder contents as an HTML table
Application: Virtual Program Office
Application Version: 2.1
Developed by (Gov't Activity): SPAWAR 08
Point of Contact Name: Joe Smith
Point of Contact Email: jsmith@spawar.navy.mil
Change History:
CHANGE #  Version      DATE      DESCRIPTION OF CHANGE
      0      1.0      15 Sep 2001      Initial release
      1      1.1      30 Sep 2001      Updated to include file size information
*****
-->
```

The following notes indicate differences between the stylesheet and schema header only.

Header Item	Description
<i>Stylesheet Name:</i>	The standard name of the <a href="#">schemastylesheet</a> file. See <a href="#">Document Naming Convention</a>
<i>Stylesheet Version:</i>	The version of the schema. Use a major and minor version number separated by a '.'

## Initial XML Developer's Guide - 29 October 2001 Appendix F

<i>Tested to:</i>	Information regarding the specific schema and software this stylesheet has been tested with.
**** Schema Name:	Name(s) of the schemas this stylesheet has been tested with.
**** Schema Version:	Version(s) of the schemas this stylesheet has been tested with.
**** XSL Processors:	Name(s) of the XSL processors this stylesheet has been tested with.
<i>Other Stylesheets Imported</i>	Stylesheets, like schema can be constructed modularly. Provide information here regarding other stylesheets reused.
<i>The next two fields are repeatable</i>	
**** File Name:	The standard name of the file. See <a href="#">Document Naming Convention</a>
*** Available at (URL):	If this Stylesheet is <a href="#">URL</a> accessible, put its address here.

## Sample Instance header

It is important that XML [documents](#) include some basic information. Most of the needed information can be gleaned from the header data provided by the [schema](#) that describes the document and the [stylesheet](#)(s) that transform or render it. The [XML](#) specifications provide syntax for pointing to schemas and stylesheets at the beginning of an XML document. In cases where validation against a schema and/or transformation with a stylesheet is not required, it is still desirable to provide references to schemas and stylesheets if available, consider this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!--
    Schema and Stylesheet Reference Data:
    stylesheet type = xslt
        url = http://spawar.navy.mil/stylesheets/SPAWARVPO$2-
1_ViewFolderData$1-1.xsl
        version = 1.1
    schema type = XML Schema (W3C)
        url = http://spawar.navy.mil/schemas/SPAWARVPOV2-
1FolderDataV1-1.xsd
        version = 1.1
  -->
  <root />
```

## Appendix G – Draft Glossary and Acronyms

The following draft glossary is provided in advance of the DONXML TechTeam's future XML Glossary deliverable. It represents the understanding and opinion of the editor, and does not reflect the consensus of the DONXML WG. These items are provided for information only.

### Terms

**Abstract** – In the context of an [XML Schema](#), an XML [element](#) or [Schema type](#) may be declared abstract, meaning that it may not be used directly. An abstract element may not be directly used in an instance, but must have in its [substitution group](#) a non-abstract element. For instance, an abstract element, 'Address', which defines the contents of an address. A non-abstract 'HomeAddress' element that is substitutable for 'Address' can be used as an XML element. The 'HomeAddress' structure reuses the previously defined 'Address' contents, but the tag provides a specific context. Schema types may also be declared abstract, similar to abstract elements, abstract types may not be directly used to reference elements, but must have a non-abstract type that extends/restricts from it. The non-abstract type can then be used to reference XML elements. The concept of abstractness is taken from object-oriented programming, where an abstract [class](#) may be defined; requiring sub-typing prior to instantiation.

**Binding** - A term frequently used in reference to XML applications taken from the field of computer science. In the context of applications that have a public interface that communicates in XML (such as the case with a [web service](#)), binding refers to the information required and the process by which an external source connects to, and interacts with it to get data in XML. Binding can also refer to the process and application required to connect a software module (e.g. a [Java class](#), or [COM object](#)) to a [public XML interface](#) or the way in which the public XML is related to an underlying data source (such as a relational database).

**BPSS** - The [Business Process Specification Schema](#) was developed as part of the ebXML project as a [schema](#) for describing a business process in an XML [instance](#). It may be created from UML models of business processes developed according to the [UMM](#) as described in the technical report, [Business Process and Business Information Analysis Overview v1.0<sup>xx</sup>](#). The BPSS is available in either [DTD format](#)<sup>xxi</sup> or [XML Schema \(Candidate Recommendation\) format](#)<sup>xxii</sup>.

**Business Term** - The [ebXML specifications](#) refers to a *business term* as a commonly used term referencing a commonly understood concept within a specific domain. To enhance understandability, it is appropriate to use business terms as XML Element names (when they exist), rather than the often esoteric [ISO 11179](#) syntax.

**C4ISR** – Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance

**Camel Case** – A convention in which names of [elements](#) and [attributes](#) are all lower case with the exception of the beginning of a new word, which is in uppercase. ebXML differentiates between *upper* camel case where the first letter of the name is also capitalized and *lower* camel case where it is not. Example of an upper camel case name: UpperCamelCase. A lower or just camel case name: lowerCamelCase. Camel case is emerging as the industry norm for XML element naming. [ebXML](#) specifies elements to be in upper and attributes to be in lower camel case, while [BizTalk](#), [RosettaNet](#), and [Oasis](#) use straight camel case for both elements and attributes.

**CSS** - [Cascading Style Sheets](#). A set of W3C recommendations for styling HTML and XML documents based on the application of formatting instructions in a linear, cascading fashion. CSS is an alternative to styling XML with [XSL](#), but CSS does not have the transformational component of [XSLT](#).

**Class** – A software component that provides instructions for the creation of an object. Applications are said to create *instances* of a class ("[objects](#)") through a process referred to as *instantiation*. In the context of XML, a [schema](#) is a "class" that describes XML [instances](#) ([data "objects"](#)).

**COE XML Registry** – The [COE XML Registry](#)<sup>xxiii</sup> "...provides a baseline set of XML components developed through coordination and approval among the COE community. The Registry allows you to browse, search, and retrieve data that satisfy your requirements." DON XML Policy requires that all



## Initial XML Developer's Guide - 29 October 2001 Appendix G

activities developing XML in the DON register components developed with the appropriate [COE XML Namespace](#).

**COE Namespace** – The COE XML Registry is divided into a “Namespaces”. “A Namespace is a collection of people, agencies, activities, and system builders who share an interest in a particular problem domain or practical application. This implies a common worldview as well as common abstractions, common data representations, and common metadata. The COE Data Emporium, including the XML Registry, allows Namespaces to publish their existence and their available information resources so that outsiders may discover them and assess whether or not they want to share.” A COE XML Namespace is an extension of the XML Namespace concept. The terms “[XML Namespace](#)” and “COE XML Namespace” are not synonymous.

**COE Namespace Manager** – Each COE XML Namespace has a central activity responsible for it. The individual responsible for coordinating and administering the Namespace is the Namespace manager. Point of contact information for the Namespace Managers is available by clicking on the [Namespace hyperlinks](#)<sup>xxiv</sup> on the registries web site.

**COE XML Namespace Prefix** – Each COE XML Namespace has been assigned a three-letter prefix that may be used as XML Namespace qualifiers in XML instances and Schemas.

**COE XML Registration Package** – Activities developing XML within the DON are required to submit a specially formatted package of information to the COE Registry containing metadata about the components registered. Information about how and what to register can be found [here](#)<sup>xxv</sup>.

**COM Object** – The Common Object Model is a Microsoft sponsored interface specification for creating interoperable software components. Distributed COM or DCOM is Microsoft's COM interface standard for distributed computing, i.e., where an “application” consists of software “objects” distributed across nodes of a network. DCOM is similar to the Java based EJB specification, but works only for Microsoft operating systems. DCOM objects can communicate via TCP/IP and their own proprietary messaging framework (Windows Distributed iNternet Architecture or DNA). Alternatively, COM objects can communicate with other non-COM / non-Windows objects such as Java [Classes](#) or [EJB's](#) via XML and SOAP.

**CORBA** – Common Object Request Broker Architecture. CORBA is a framework created by the [Object Management Group](#)<sup>xxvi</sup> (OMG) to facilitate platform / operating system / programming language neutral distributed computing. Software components or “objects” interact in a client-server relationships, with an Object Request Broker (ORB) software component acting as intermediary. Via the [IIOP](#), CORBA based distributed applications can operate across the Internet. Most commonly used with the Java language, though CORBA is language independent.

**Core Components** – One goal of the ebXML effort is to define a set of universal, [core components](#) that are contextually neutral and can be used across all domains to express semantics of common business concepts. Core components may be information entities, defined in the ebXML Core Component Dictionary [technical reports](#), or process components discussed in the ebXML Business Process technical reports. Note that the core component technical reports do not address how an information component will be expressed in XML – this was an intentional omission on the part of ebXML. It was felt that prior to defining rules for creation of XML, a necessary first step was to create a [schema](#) neutral standard for defining components in business terminology. The work of defining how core components map to XML will be undertaken by the [Core Component Project Team](#)<sup>xxvii</sup> of the UN/CEFACT sponsored [Electronic Business Transition Working Group](#) (eBTWG).

**DDDS** – The [Defense Data Dictionary System](#)<sup>xxviii</sup> defines standard data elements per the [DOD 8320 series of documents](#)<sup>xxix</sup>. The DDDS provides definitions of Standard Data Elements ([SDEs](#)) from core data models across all DOD data domains. The DDDS elements are mainly logical in nature, and may be used to express logical, semantic relationships between [XML elements](#). [XML Schema types](#) may be used to express relationships to DDDS standard data elements.

**Document Type Declaration** – A declaration at the beginning of an [XML document](#) indicating a [DTD](#) to which the [instance](#) must conform.

**DOM** - The Document Object Model. The [set of W3C DOM recommendations](#)<sup>xxx</sup> form application [interface descriptions](#) ([APIs](#)) for expressing the contents of XML or HTML “documents” as hierarchical tree-like

**Initial XML Developer's Guide - 29 October 2001 Appendix G**

models of information with data forming the “leaves” of the tree. [XML Processors](#) that implement the DOM interface [parse](#) an entire [XML document](#), creating a data tree in memory. Applications that call a DOM parser access data from the XML object tree through a set of programmatic instructions defined by the specifications. The instructions allow applications to “walk the document tree”, searching for elements and attributes that meet query criteria ([XPath](#) expressions). Results are returned to the calling application and assigned to application variables for further processing.

**DTD** - Document Type Definition. A [schema](#) syntax that is part of the [XML](#) 1.0 specification and derived from [SGML](#).

**EJB** – Enterprise Java Beans. EJB is an [interface](#) specification which a Java [class](#) may implement. Software objects that implement the EJB interface may interoperate in an enterprise (distributed) environment, even across the Internet via TCP/IP and the [CORBA IIOP](#). In this fashion, an “application” may consist of a number of independent software components (“[objects](#)”) that are physically separated at different nodes of a network, but functioning together as a single application similar to the Microsoft (D)[COM](#) specification.

**Entity** – In the context of a [DTD](#), an entity is a declarative construct defining, referencing text, or a binary file. Entities are defined in the DTD, and referenced elsewhere in the DTD (*parameter entity*) or in the body of the XML (*general entity*). A [validating parser](#) encountering a reference to a previously defined entity during the validation process will insert the entity’s value in place of the entity reference. *Internal entities* are declared in the DTD and may be general or parameter. External entities point to an external file containing the entity declaration via [URI](#) reference; they also may be internal or external. A parsed entity is some form of encoded text and is therefore processed by a parser. An unparsed entity is a reference to binary file that will not be parsed. Unparsed entities are always external. Through entities, DTD’s may declare a common construct once, and reuse it many times throughout the DTD or in the instance. A common use for parameter entities is to declare a common set of [attributes](#) in the DTD. Assigning the attributes to an [element](#) only requires a reference to the parameter entity, vice retyping the entire attribute list many times. A second use of external unparsed general entities is to make reference to a binary file (such as an image or sound file) within an XML instance.

**EDI** – Electronic Data Interchange. A term referring to the conduct of eBusiness through the exchange of electronic messages. Two message standards exist as rigorously defined sets and segments, one maintained by the U.S. led ANSI X12 body, and the second led by UN/EDIFACT.

**Fatal Error** - [From the XML 1.0 specification] *“An error which a conforming XML [parser](#) must detect and report to the application. After encountering a fatal error, the parser may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document’s logical structure to the application in the normal way).”* In other words, upon detecting a fatal error (such as a [well-formedness](#) violation), the parser is unable to provide information from the XML [document](#) to the calling application such that the application may continue functioning normally.

**HTML** - [Hypertext Markup Language](#)<sup>xxxi</sup>

**Interface** – The process by which a software application interacts with other software or users. In object-oriented programming an (software) “object’s” interface is often described separately from the internal logic in a process known as “encapsulation”. Essentially the interface encapsulates and hides the internal logic. This allows flexibility to change and improve object code without affecting other objects. An interface description is made public so other objects/applications know how to interact. Software is said to “implement” an interface if it conforms to the behavior as defined in an interface description. The [Object Management Group](#) (OMG) has defined a formal syntax (language) for defining interfaces in a programming language neutral fashion. This is called the [OMG Interface Description Language](#)<sup>xxxii</sup> (OMG IDL). This IDL is used to define interface specifications such as the [DOM API](#) and [CORBA](#). For developers implementing [public XML interfaces](#), it is a good idea to document exactly how other applications connect, query, and receive (i.e. [bind](#) to) your application; while it is not necessary to go to

## Initial XML Developer's Guide - 29 October 2001 Appendix G

the trouble of writing a formal IDL interface description, some kind of formal document will greatly aid other applications desiring to share data.

**IIOB** – Internet Inter-ORB Protocol. A TCP/IP based protocol that facilitates communication between [CORBA](#) ORBs. Via IIOB, CORBA client objects at one location on the Internet can communicate with CORBA server objects at another node and vice versa.

**ISO 11179** - *Information Technology - Specification and Standardization of Data Elements* is a 6-part ISO standard providing a framework and methodologies for developing, documenting, and registering standard data elements. Of interest to XML developers is Part 5: *Naming And Identification Principles For Data Elements* upon which the ebXML naming convention is based. The specifications are available from the [ISO Store](#)<sup>xxxiii</sup> under section 35.040 - Character Sets And Information Coding for a small fee.

**Markup** - Special characters used by Markup Languages ([SGML](#), [XML](#), [HTML](#)) to differentiate data from metadata. SGML allows document authors the flexibility of specifying which characters are used for markup, where as in XML the markup characters are fixed. Markup characters may not be used in data text (unless special precautions are taken). In the [tags](#) definition example, the markup characters are '<' (greater than), '>' (less than), and '/' (forward slash). The [XML specification](#)<sup>xxxiv</sup> defines start [tag](#) markup as opening with a '<' and ending with a '>'. It specifies that end [tag](#) markup as opening with '</' and ends with '>'.

**Metadata** - Data about data. For example, for the data '3000N', the metadata might be 'latitude'. [Markup](#) languages such as [SGML](#) and [XML](#) encapsulate data with [tags](#) that contain text describing the metadata. See the example provided in the [tags](#) definition.

**Normative** – A term frequently used by software specifications to mean required, mandatory, or representing the only way to accomplish something. Often references are cited as normative, meaning that the requirements of these references apply to the document being read, or as non-normative, meaning they are provided as information only.

**Object** – A term used frequently in relation to XML and computer science. Strictly speaking, an object is a run-time software construct that resides in the Random Access Memory (RAM) of the host computer. Objects are created by applications from code that defines the object's behavior; this code is called a [class](#). In object-oriented programs, objects interact with other objects to create the behavior of the application. An object's behavior is described by an [Interface](#) consisting of *methods* and *properties*. A method can be thought of as a behavior of the object that can be triggered by calling it and optionally passing parameters. For instance, the object '**myAccount**' might have the method '**getBalance(accountNumber)**'. Object oriented languages use the 'dot' notation to refer to objects and methods. From the previous example, '**currentBalance == myAccount.getBalance(accountNumber)**' is a code snippet that assigns to the '**currentBalance**' variable the balance returned from the '**myAccount**' object when the '**getBalance()**' method is called by passing in the '**accountNumber**' variable. Object properties are similar to methods, but instead of calling a behavior, a property call to an object returns a previously set value of the property. Returning to the example, '**myName == myAccount.accountOwner**' sets the '**myName**' variable equal to the '**accountOwner**' property of the '**myAccount**' object, conversely '**myAccount.accountOwner == myName**' sets the '**accountOwner**' property of the '**myAccount**' object to the value of the '**myName**' variable. XML that has been [parsed](#) by an [XML processor](#) implementing the [DOM API](#) is transformed into a set of objects that may be used by the calling application to extract data from the XML. Also, an application may construct a DOM tree of objects in memory then transmit the data to another application or object as a textually encoded string of XML. The receiving object then accesses the data via the DOM or [SAX APIs](#). Since the XML format is neutral, a [COM](#) object created by a Windows application may interact with an [EJB](#) object running on a Unix platform for true cross-platform, language independent distributed computing.

**Payload (XML)** – Protocols and frameworks such as [SOAP](#), [BizTalk](#), and [ebXML](#) use XML to markup message header information necessary for [binding](#), reliable messaging, and security. The term '*payload*' refers to the XML being transmitted that contains the actual business information communicated.

**Public (XML) Interface** – XML may be employed internal to an application or it may be used to communicate information to another systems outside the originating applications environment. The term '*Public Interface*' refers to XML used by an application or set of homogeneous applications to

**Initial XML Developer's Guide - 29 October 2001 Appendix G**

communicate with other applications across system boundaries. DOD and DON policy for registration of XML components applies to public interfaces; these policies are not intended to restrict the use of XML internal to systems; in fact, it is recommended that applications separate internal XML grammars processed by application code from that used for external communications.

**Qualified (elements and attributes)** – The practice of prefixing an element or an attribute with a [XML Namespace](#) qualifier in accordance with the [Namespaces in XML<sup>xxxv</sup>](#) [W3C Recommendation](#). This allows two elements with the same name to be disambiguated by an [XML processor](#).

**Regular Expression** – A language for defining patterns in strings and numbers. The XML Schema language allows [elements](#) and [attributes](#) to be constrained by regular expressions to provide a precise description of the range of possible values. For instance an element of type='integer' could be further constrained to be only a 3 digit integer by the regular expression '/d{3}'.

**Rendering (XML)** - XML is not easily useable to readers in its native format and should be transformed for presentation (rendered), rendered for presentation, either by a [CSS](#), [XSLT](#) (to [well-formed HTML](#)) for browser viewing, or by [XSL-FO](#) into a format for viewing by another presentation applications (e.g. into Adobe Acrobat .pdf, or MS Word .doc files.) Note: It is a common assumption that all XML must be rendered (by a [stylesheet](#)) to be useful therefore all XML must have a stylesheet. This is a mistake; XML data can be used by an application via an [API](#) and never get rendered at all.

**SAX** - Simple [API](#) for XML. [SAX<sup>xxxvi</sup>](#) is an open-source interface for accessing information from XML [documents](#). SAX [parsers](#) process a document, triggering events in the calling application corresponding to the parser encountering opening [tags](#), closing tags and character data. Accessing XML data via SAX is very quick and places less demands on system resources than [DOM](#), however once processed, a document must be re-parsed if the required information was not retained initially. This can be conceptualized as "serial" access to the information.

**Schema** - Within the context of XML, a document describing a set of XML Instances. Schemas may be expressed in a number of different languages. Most familiar is the Document Type Definition ([DTD](#)) syntax described in the [XML 1.0](#) specification. Schemas provide the rules against which a [validating parser](#) [validates](#) an [instance](#) of XML.

**SGML** - The Standard Generalized Markup Language [[ISO 8879<sup>xxxvii</sup>](#)]. SGML is the parent of both [HTML](#) and [XML](#).

**SOAP** - "SOAP is the Simple Object Access Protocol, a way to create widely distributed, complex computing environments that run over the Internet using existing Internet infrastructure. SOAP is about applications communicating directly with each other over the Internet in a very rich way." [MS] "SOAP is a protocol specification for invoking methods on servers, services, components, and objects. SOAP codifies the existing practice of using XML and HTTP as a method invocation mechanism. The SOAP specification mandates a small number of HTTP headers that facilitate firewall/proxy filtering. The SOAP specification also mandates an XML vocabulary that is used for representing method parameters, return values, and exceptions." [DevelopMentor]. Taken from the [XML Cover Pages<sup>xxxviii</sup>](#). The current [SOAP 1.1 specification<sup>xxxix</sup>](#) is a [W3C Note](#); [SOAP 1.2<sup>xl</sup>](#) is going through the W3C [consensus process<sup>xli</sup>](#) and was published as a first working draft in July 2001.

**SQL** - Structured Query Language - A language for querying, writing to, and constructing relational databases. Many versions of SQL exist; meaning that an SQL query that works for one database will not necessarily work against another.

**SDE** – Standard Data Element as defined by the DOD 8320 series and used in the [DDDS](#).

**Stylesheet** - A generic term that may refer to an [XSL Stylesheet](#) or a [CSS](#). Often the term used to reference XSL Stylesheets implicitly, however this is not technically correct as a stylesheet may be CSS conformant, and having nothing to do with XML whatsoever. The primary function of a stylesheet is to [render](#) XML to a presentation format. However, [XSLT](#) can transform one XML [instance](#) into another different [instance](#). Application of a stylesheet by an [XSL processor](#) to an XML [document](#) for the purpose of creating another XML [document](#) (i.e. an XML to XML transformation) does not render a presentation format at all. More simply, applying a stylesheet to XML doesn't imply that the output is ready for viewing; you have to understand what the stylesheet is doing.



## Initial XML Developer's Guide - 29 October 2001 Appendix G

**Substitution Group** – In the context of [XML Schemas](#), a substitution group may be declared for an [element](#) to define a synonymous group of [tag](#) names. A top-level element is declared, then other elements are declared with an attribute indicating they belong in the substitution group of the top element. Different elements do not necessarily have to have the same structures – used in this fashion they are functionally similar to a group of optional elements where only one may be chosen. The top-level element may be declared [abstract](#), in this case the top level element may not be used but can serve as a generic model for non-abstract elements in the substitution group. This is similar and somewhat redundant of the functionality provided by [XML Schema types](#).

**Throw (an error)** – A terms adopted from the Java language to indicate that a processing error has occurred. Conceptually, Java “throws” the error to an error-handling [object](#), which “catches” it, or may “throw” it to another object, and so on.

**UID** – Unique Identifier. A generic term used to indicate that an object or item has a string or number that identifies it uniquely within a specific context or environment. Universally Unique Identifiers (UUIDs) and Globally Unique Identifiers (GUIDs) are special identifiers that are guaranteed universal uniqueness via an identifier assignment algorithm.

**UML** - The [Unified Modeling Language](#)<sup>xlii</sup> defines a standard language and graphical notation for creating models of business and technical systems. UML is not only for programmers, it defines several model types that span a range from functional requirements definition and activity work-flow (business process) models to logical and physical software design and deployment. The UML has over the last few years become the *lingua franca* for business and technical stakeholders to communicate and develop IT systems. Through the [UMM](#), UML has been adopted by [UN/CEFACT](#) and [ebXML](#) as the modeling language of choice.

**UMM** - The [Unified Modeling Methodology](#)<sup>xliii</sup> is a product of [UN/CEFACT](#), and describes the CEFACT recommended methodology for modeling business processes to support the development of the next generation EDI. It is based upon the [Rational Unified Process](#)<sup>xliiv</sup>, and uses the [UML](#) as it modeling language. In the UMM, business process are modeled by deconstructing them into a series of [document](#) exchanges which are orchestrated to form a complex process. The ebXML Technical Report, [Business Process and Business Information Analysis Overview v1.0](#) further develops the UMM. The ebXML [Business Process Specification Schema v1.01](#) (BPSS) provides a [schema](#) in the form of a [DTD](#) for specifying business processes as an [XML instance](#), it may be developed as part of a UMM modeling process.

**URL / URI / URN** – Uniform Resource Locators, Uniform Resource Indicators, and Uniform Resource Names are different, related methods of uniformly referencing resources across networked environments. A recently release [W3C Note explains the difference](#)<sup>xliv</sup>.

**Valid (XML)** - An XML [instance](#) ([document](#)) whose structure has been verified in conformance to a [schema](#) by a [validating parser](#). Note that an XML [instance](#) must be [well-formed](#) to be valid, but it does not need to be valid to be well-formed. This is because a parser will always check well-formedness constraints but will only checking validation constraints if it is a [validating parser](#).

**Validating Parser** - An XML [parser](#) that enforces [validity](#) constraints by comparing the structure and syntax of an XML [instance](#) to the rules specified in a [schema](#). Not all parsers are validating parsers, and validating parsers enforce validation according to specific schema languages. Most validating parsers are capable of enforcing validity against a [DTD](#), while some can enforce validation rules described in other schema languages.

**W3C** - The [World Wide Web Consortium](#)<sup>xlvi</sup> was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has more than 500 [Member organizations](#)<sup>xlvii</sup> from around the world and has earned international recognition for its contributions to the growth of the Web.

**W3C Recommendation** - A work that represents [consensus](#)<sup>xlviii</sup> within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission.

## Initial XML Developer's Guide - 29 October 2001 Appendix G

**W3C Note** – A W3C Note is a publication of a member idea. Notes do not go through the consensus process, they represent the ideas of a single (group of) W3C member(s).

**(W3C) XML Schema** - A [schema](#) written in according the W3C XML Schema language. [From the [W3C Schema](#)<sup>xlix</sup> page] "XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. The [XML Activity Statement](#)<sup>i</sup> explains the W3C's work on this topic in more detail." The W3C XML Schema language is described in three [recommendations](#): [XML Schema Part 0: Primer](#)<sup>ii</sup>, [XML Schema Part 1: Structures](#)<sup>iii</sup>, and [XML Schema Part 2: Datatypes](#)<sup>iiii</sup>. In the DONXML Developers Guidance (this document), the term *XML Schema* will be used in reference to a W3C XML Schema language compliant [schema](#).

**Web-service** – A generic term used to refer to the use of Hypertext Transfer Protocol (HTTP) and XML to exchange information. Frequently the term implies the use of [SOAP](#) to exchange information between applications, vice application to human, which is done in [HTML](#).

**Well-formed (XML)** - An XML [instance](#) that meets well-formedness constraints defined by the [XML 1.0](#) specification. Well-formedness constraints are precise syntactic rules for [markup](#) of data. As an example, the XML specification stipulates every open [tag](#) must have a corresponding and properly nested closing [tag](#). A [document](#) must be well-formed in order to be considered XML. A parser processing a [document](#) will [throw](#) a [fatal error](#) if it detects a well-formedness violation.

**Well-formed HTML** - HTML that meets the [well-formedness](#) constraints of [XML 1.0](#). Well-formed HTML is not the same as [XHTML](#).

**XHTML** - [Extensible HyperText Markup Language](#)<sup>liv</sup>.

**XML** - [From [the XML 1.0 specification](#)] "*Extensible Markup Language, abbreviated XML, describes a class of data objects called [XML documents](#) and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML. By construction, [XML documents](#) are conforming [SGML](#) documents.*" The XML 1.0 specification is a [W3C Recommendation](#). In XML, [metadata](#) is described by an extensible set of [tags](#); the tags are said to be extensible, because unlike [HTML](#), where the [markup](#) tags are fixed, developers are given the flexibility to define their own tags or reuse tags defined by another party. This flexibility is both the key to XML's power and the single biggest stumbling point to achieving interoperability when making use of XML.

**(XML) API** - Application Programming Interface. In the context of XML, [parsers](#) expose their data to a calling application via an interface. An interface is a specification (which the parser conforms to) that describes how the parser will pass data from an XML [document](#) to a calling application. The two accepted XML API's are [DOM](#) and [SAX](#).

**(XML) Attributes** – In the context of XML, attributes provide a mechanism for attaching additional metadata to an [XML element](#). For example, <element attribute="value"/>. An XML attribute is not equivalent to an object or relational model attribute. Data model entity attributes may be expressed as either XML attributes or elements. Frequently in discussions surrounding the application of XML to data models, one party will be referring to attributes in the context of XML and another to attributes in the context of data models, causing confusion.

**XML Comments** – The structure for inserting free text comments into XML. The same structure is used for [SGML](#) and [HTML](#) comments. <!-- comment text here -->

**XML Component** – A generic term used to refer to [XML elements](#), [attributes](#), and [XML Schema type](#) definitions.

**(XML) Document** - - [Paraphrased from the XML 1.0 specification] "*A data object is an XML document if it is [well-formed](#), as defined in the [XML 1.0](#), specification. A [well-formed](#) XML document may in addition be [valid](#) if it meets certain constraints*" as described by a [schema](#). Synonymous with XML [instance](#).

**(XML) Elements** – The fundamental unit of information in XML. Elements are encapsulated by [tags](#), and may contain (among other things) [attributes](#) (declared inside the opening tag), other elements, or data.

## Initial XML Developer's Guide - 29 October 2001 Appendix G

**(XML) Child Element** – The hierarchical nature of XML allows [elements](#) to contain or be nested inside other elements, forming a conceptual data tree (see [DOM](#)). Often XML elements are referenced in terms of parent-child relationships. A child element is an element contained between the [tags](#) of a parent element. Child elements are also referred to as *descendants*, while parent elements may be referred to as *ancestors*.

**(XML) Grammar / Vocabulary** – Related terms often used synonymously to indicate a set of [element](#) and [attribute](#) names and the structures described by a [schema](#) or set of related [schemas](#) that employ the elements and attributes. More precisely, the term vocabulary implies a commonly defined set of elements and attributes, while grammar refers to the composition of the vocabulary into meaningful business documents by one or more related schemas. An [XML Namespace](#) may be used to describe a vocabulary, while a schema may employ vocabulary from a single or multiple XML Namespaces.

**(XML) Instance** - Synonymous with [XML Document](#). The term derives from object-oriented programming where objects are considered instances of classes. Programmers write code that defines application behavior in terms of classes of objects. In application execution, objects are *instantiated* (see [object](#)) from these class definitions. XML provides an object-like way to conceptualize textual data. Essentially, [schemas](#) are the equivalent of object classes, and XML [documents](#) are equivalent of object instances. Hence the term XML instance is widely used, however XML [document](#) is the official term used by the W3C.

**XML Namespace** – An XML Namespace is a conceptual “space” to which [element](#) and [attribute](#) names may be assigned. An XML Namespace is declared within an XML [instance](#) by assigning a [URI](#) reference and an optional [qualification](#) prefix to an element. The element and all its children are considered to be “in” the XML Namespace unless specifically qualified with another Namespace’s prefix. The URI reference does not have to an associated document physically at the URI. Within an [XML Schema](#), the ‘targetNamespace’ attribute may be used to indicate that all elements declared within the schema are to be treated as “in” the target Namespace. The W3C [Recommendation Namespaces in XML](#)<sup>iv</sup> provides the full specification for XML Namespaces. Note: [COE XML Namespaces](#) may use XML Namespaces but the two terms are not synonymous.

**(XML) Name Token** – Per the [XML 1.0](#) specification, a Name Token is “...any mixture of name characters...” where a “name” character obey the XML name convention. A [XML] *Name* “...is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string “xml”, or any string which would match (('X'|'x') ('M'|'m') ('L'|'l'))”, are reserved for standardization in this or future versions of this specification.” White space characters (hex #x20, #x9, #xD, #xA) are excluded from Name Tokens.

**(XML) Parser** - A software application (module) that either reads or receives a text encoded binary stream, decodes it, verifies the input conforms to “[well-formedness](#)” constraints of the [XML 1.0](#) specification, (in the case of a Validating Parser) checks [validity](#) of the [XML Instance](#) against a [schema](#) if available, and exposes the content via an [API](#) to a calling application. A parser can be a standalone application, but it is most often a module called by a larger program (the calling application). A Parser may also be referred to as an [XML Processor](#).

**(XML) Processor** - A synonym for an XML [parser](#).

**XML Declaration** – Every well-formed XML [document](#) must begin with a statement that as a minimum declares the version of [XML](#) that the document conforms to. Example: <?xml version=”1.0”>.

**XML Document Tree** – Refers to the logical model of an XML [document](#) conceptualized as a data tree, with a [Root Node](#) and branch nodes ending at data that can be thought of as the leaves. See [DOM](#).

**(XML) Root Node** – The first node originating the XML [Document Tree](#). The Root Node is not the same as the [root element](#).

**(XML) Root Element** – Refers to the [XML element](#) in which all other elements must be nested. The root element (a physical XML construct) is a child of the logical [root node](#) of the [document tree](#).

## Initial XML Developer's Guide - 29 October 2001 Appendix G

**(XML Schema) Type** – An [XML component](#) defined by the [XML Schema](#) language. Types do not show up in XML [instances](#); they are used within the Schema to express relationships, and through type inheritance, add an object-like capability to XML Schemas. Types may be simple, that is they allow definition of simple data-type constraints on element values; or they may be complex, that is they define structures consisting of other elements. For example a type could be defined `<xsd:complexType name="AddressDetails">...</xsd:complexType>`, then the definitions for [XML elements](#), *'ShippingAddress'* and *'MailingAddress'* could reference the previously defined generic type.

**(XML) Schema Annotation** – The XML Schema language allows addition of annotations to schema components through an *'annotation'* element (`<xsd:annotation>`) which must contain either a *'documentation'* element (`<xsd:documentation>`) or *'AppInfo'* element (`<xsd:appInfo>`). A *'source'* attribute may be added to either element to provide a URL reference to the source of the annotation. Annotations provide a more sophisticated way to provide documentation and application information that may be [parsed](#) and accessed by applications via an [API](#).

**(XML) Tags** - [XML](#) (and its parent [SGML](#)) annotate [metadata](#) through the use of tags that indicate which text in a [document](#) are considered metadata and which is to be considered data. Tags are surrounded by [markup](#) characters. As an example, the data '3000N' can be marked up in XML, `<latitude>3000N</latitude>`. The tags are `<latitude>` (start tag) and `</latitude>` (end tag). Note: As discussed in the [XML](#) definition presented here, developers are free to define tags. As an example, the data '3000N' could be alternatively marked up as, `<lat>3000N</lat>`, and still be [well-formed](#). The [document schema](#) will specify which of all possible well-formed XML [instances](#) are [valid](#) for a particular application. An additional example is `<Latitude hemisphere="N">3000</Latitude>`; here the tag contains an [XML attribute](#) to specify the hemisphere. The choice as to the attribute name and possible values are also at the developer's discretion. Note that [Parsers](#) processing [documents](#) are sensitive to markup tag case, therefore in the first example the tag `<latitude>` is not equivalent to the later example tag, `<Latitude>`.

**XPath** – [XPath](#) is a W3C recommendation whose primary purpose is to provide a compact, non-XML notation for identifying parts of an XML document. It operates on the abstract, logical structure of an XML document, rather than its surface syntax by modeling an XML document as a tree of nodes. The document tree can be navigated by applications implementing XPath. XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [[XSLT](#)] and XPointer.

**XSL** - The Extensible Style Sheet Language. [From the [W3C XSL page](#)<sup>[vi]</sup>] "XSL is a language for expressing stylesheets. It consists of three parts: [XSL Transformations](#)<sup>[vii]</sup> (XSLT): a language for transforming XML documents, the [XML Path Language](#)<sup>[viii]</sup> (XPath), an expression language used by XSLT to access or refer to parts of an XML document (XPath is also used by the [XML Linking](#)<sup>[ix]</sup> specification). The third part is [XSL Formatting Objects](#): an XML vocabulary for specifying formatting semantics. An XSL stylesheet specifies the presentation of a class of XML documents by describing how an [instance](#) of the class is transformed into an XML document that uses the formatting vocabulary. For a more detailed explanation of how XSL works, see the [What Is XSL](#)<sup>[x]</sup> page." As of 16 October 2001, [XSL](#)<sup>[xi]</sup> is a W3C final [recommendation](#).

**XSL Processor** - The software (module) executing XSL transformation and formatting instructions. At a minimum, consists of an [XSLT](#) conformant transformation component, and an optional [XSL-FO](#) processing component. A word of caution: XSL processor vendors often add "extensions" to the [XSLT](#) specification. While often extremely useful, stylesheets written using these extensions will not perform correctly in another XSLT compliant processor, eliminating their cross-platform compatibility.

**XSL-FO** - XSL Formatting Objects: an XML vocabulary for specifying formatting semantics. XSL-FO works in conjunction with [XSLT](#) to markup transformed XML with formatting object [tags](#). Applications capable of processing these tags [render](#) the XML to another application's presentation environment. For example, Apache's Formatting Object Processor ([FOP](#)) can transform XML to Adobe PDF format. Another example is [ifor](#), an open-source formatting object processor for transforming XML to Rich Text Format (RTF).



## Initial XML Developer's Guide - 29 October 2001 Appendix G

**XSLT** - [XSL Transformations](#)<sup>lxii</sup>, a [W3C recommendation](#) [from the XSLT recommendation] "...defines the syntax and semantics ... for transforming XML documents into other XML documents" [including [well-formed HTML](#)]." XSLT is the only W3C recommended XML syntax for transforming XML [documents](#). Developers writing stylesheets should ensure they are strictly conformant to this specification to ensure reusability. Conformance testing through the use of several XSLT compliant [XSL processors](#) is recommended.

---

## URL References

- <sup>i</sup> Navy XML Quick Place, <http://quickplace.hq.navy.mil/navyxml>
- <sup>ii</sup> Task Force Web, <http://www.tfw.navy.mil/>
- <sup>iii</sup> RFC 2119, <http://www.ietf.org/rfc/rfc2119.txt>
- <sup>iv</sup> XML Schema Tutorial, <http://www.xfront.com/xml-schema.html>
- <sup>v</sup> XFront.com, <http://www.xfront.com/>
- <sup>vi</sup> Schema Best Practices, <http://www.xfront.com/BestPracticesHomepage.html>
- <sup>vii</sup> eBTWG, <http://www.ebtwg.org/>
- <sup>viii</sup> eBTWG UML2XML, <http://www.ebtwg.org/projects/u2xdr.html>
- <sup>ix</sup> COE XML Registry, <http://diides.ncr.disa.mil/xmlreg/user/index.cfm>
- <sup>x</sup> ebXML Specifications and Technical Reports, <http://www.ebxml.org/specs/>
- <sup>xi</sup> COE Data Emporium, <http://diides.ncr.disa.mil/shade/refdatasets.cfm>
- <sup>xii</sup> MIL-STD-6040 (USMTF), [http://www-usmtf.itsi.disa.mil/std\\_6040.html](http://www-usmtf.itsi.disa.mil/std_6040.html)
- <sup>xiii</sup> OASIS, <http://www.oasis-open.org/>
- <sup>xiv</sup> UN/CEFACT, <http://www.unece.org/cefact>
- <sup>xv</sup> eBTWG, <http://www.ebtwg.org/>
- <sup>xvi</sup> ebXML Core Component Technical Reports, [http://www.ebxml.org/specs/#technical\\_reports](http://www.ebxml.org/specs/#technical_reports)
- <sup>xvii</sup> ebXML Technical Architecture, <http://www.ebxml.org/specs/ebTA.pdf>
- <sup>xviii</sup> ebXML Technical Report, Naming Convention for Core Components  
<http://www.ebxml.org/specs/ebCCNAM.pdf>
- <sup>xix</sup> W3C Technical Recommendations, <http://www.w3.org/TR/>
- <sup>xx</sup> Business Process and Business Information Analysis Overview v1.0,  
<http://www.ebxml.org/specs/bpOVER.pdf>
- <sup>xxi</sup> ebXML Business Process Specification DTD, <http://www.ebxml.org/specs/ebBPSS.dtd>
- <sup>xxii</sup> ebXML Business Process Specification XML Schema (CR), <http://www.ebxml.org/specs/ebBPSS.xsd>
- <sup>xxiii</sup> COE XML Registry, <http://diides.ncr.disa.mil/xmlreg/user/index.cfm>
- <sup>xxiv</sup> COE XML Namespace Managers, [http://diides.ncr.disa.mil/xmlreg/user/namespace\\_list.cfm](http://diides.ncr.disa.mil/xmlreg/user/namespace_list.cfm)

- xxv COE XML Registration Information, [http://diides.ncr.disa.mil/xmlreg/user/registry\\_info.cfm#submit](http://diides.ncr.disa.mil/xmlreg/user/registry_info.cfm#submit)
- xxvi The Object Management Group, <http://www.omg.org/>
- xxvii eBTWG Core Component Project, <http://www.ebtwg.org/projects/core.html>
- xxviii DDDS, <http://www-datadmn.itsi.disa.mil/ddds/ddds40.html>
- xxix DOD 8320, <http://www-datadmn.itsi.disa.mil/guidance.html>
- xxx W3C DOM, <http://www.w3.org/DOM/>
- xxxi HTML, <http://www.w3.org/MarkUp/>
- xxxii OMG IDL, [http://www.omg.org/gettingstarted/omg\\_idl.htm](http://www.omg.org/gettingstarted/omg_idl.htm)
- xxxiii ISO Store, <http://www.iso.ch/iso/en/prods-services/ISOstore/store.htm>
- xxxiv XML 1.0, <http://www.w3.org/TR/2000/REC-xml-20001006>
- xxxv Namespaces in XML, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- xxxvi SAX, <http://www.megginson.com/SAX/>
- xxxvii ISO 8879 (SGML), <http://www.w3.org/TR/2000/#ISO8879>
- xxxviii XML Cover Pages - SOAP, <http://xml.coverpages.org/soap.html>
- xxxix SOAP 1.1, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- xl SOAP 1.2, <http://www.w3.org/TR/2001/WD-soap12-20010709/>
- xli W3C Process, <http://www.w3.org/Consortium/Process-20010719/submission>
- xlii UML, <http://www.rational.com/uml/index.jsp>
- xliii Unified Modeling Methodology, <http://www.gefeg.com/tmwg/tm090.pdf>
- xliv Rational Unified Process, <http://www.rational.com/products/rup/index.jsp>
- xlv W3C Note, URI/URL/URN Clarification, <http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/>
- xlvi W3C, <http://www.w3.org/>
- xlvii W3C Members, <http://www.w3.org/Consortium/#membership>
- xlviii W3C Consensus Processes, <http://www.w3.org/Consortium/Process-20010719/submission>
- xl ix W3C Schema page, <http://www.w3.org/XML/Schema>
- l W3C Activity Statement, <http://www.w3.org/XML/Activity.html>
- li XML Schemas: Part 0, <http://www.w3.org/TR/xmlschema-0/>
- lii XML Schemas: Part 1, <http://www.w3.org/TR/xmlschema-1/>
- liii XML Schemas: Part 2, <http://www.w3.org/TR/xmlschema-2/>
- liv XHTML, <http://www.w3.org/MarkUp/#xhtml1>
- lv Namespaces in XML, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- lvi W3C XSL Page, <http://www.w3.org/Style/XSL/>
- lvii XSL Transformations, <http://www.w3.org/TR/xslt>
- lviii XPath, <http://www.w3.org/TR/xpath>
- lix XLink, <http://www.w3.org/TR/xlink/>
- lx What is XSL, <http://www.w3.org/Style/XSL/WhatIsXSL.html>

<sup>lxi</sup> XSL Final Recommendation, <http://www.w3.org/TR/2001/REC-xsl-20011015/>

<sup>lxii</sup> XSLT, <http://www.w3.org/TR/xslt>